Programming for Engineers
Fortran: Input and Output

Abu Hasan Abdullah

January 7, 2009

Course Text

Mayo W. E. and Cwiakala M. (1995): *Programming with Fortran 77*
ISBN 0-07-041155-7, McGraw-Hill

Preview

- List Directed I/O
- Formatted I/O
- Format Statement
- Edit Descriptors

List Directed I/O

- List directed I/O provide the easiest way to input and output data from a program, where
  * appearance of data is not of any concern, only what we input or output
  * computer controls all aspects of the appearance of output
  * getting quick answer is desired

- General form for input

```
READ *, variable1, variable2, . . .
```

- General form for output

```
PRINT *, variable1, variable2, . . .
```

List Directed I/O

- The asterisk (*) in both statements indicates we are using the *free format*
- `variable1`, `variable2`, . . . appearing after the `READ *` and `PRINT *` are called the *Input/Output (I/O) list*
- Show how **Example 3.2** works
- Show how **Example 3.3** works

EXAMPLE 3.2

The PRINT command shown below will send the value of each variable to the CRT screen. For example, if we enter the data given above, we can print them out using the following program segment:

```
READ *, X, Y, Z
PRINT *, X, Y, Z
```

the computer will print out the same values that you typed in:

```
14.3, -27.943, 0.0034567  <CR>              (what you type in)
14.30000  -27.94300  0.003456700   (printout on CRT screen)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
          |         |         |         |         |
         10        20        30        40        50
```

EXAMPLE 3.3

Here is an example of enhancing the PRINT statement with strings that describe the data being printed:

```
X = 2.4
PRINT *,'X = ', X,'X ** 2 = ', X*X
```

This will produce the following output on the screen:

```
X = 2.400000 X ** 2 = 5.760000
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
         |         |         |         |         |
        10        20        30        40        50
```

Formatted I/O

- *Formatted I/O* allows control over I/O functions which list-directed I/O commands, seen previously, do not.

- This introduces *I/O FORMAT pair*:

  * For input

```
      READ sl, variable1, variable2, . . .
sl    FORMAT (list of instructions)
```

  * For output

```
      PRINT sl, variable1, variable2, . . .
sl    FORMAT (list of instructions)
```

  where `sl` is a number used as the statement label. See the examples below.

Formatted I/O

- **Examples:**
  * For input

```
      READ 10, SPEED, TIME
10    FORMAT (1X,F12.3,2X,F12.3)
```

  * For output

```
      PRINT 20, DENCT, VOLUME
20    FORMAT (1X,F12.3,2X,F12.3)
```

  Note: Without matching I/O commands, the `FORMAT` statement is useless.

Formatted I/O

- Difference between list-directed I/O

```
PRINT *, ICOUNT, JCOUNT
```

  and formatted I/O

```
      PRINT 33, ICOUNT, JCOUNT
33    FORMAT (' ',I6, I9)
```

  is that the * is being replaced with a statement label (33 in the example above) pointing to a FORMAT statement

`FORMAT` Statement

- List of instructions that follows `FORMAT` statement is composed of:
  - a carriage control character (output only) to reset printer if used
  - a list of edit descriptors to specify the output instructions for each output item, such as
    * type of variable and number of significant digits
    * column in which to start printing
    * floating point or exponential form (real numbers)
    * number of blank spaces and blank lines
    * any added text to be included

FORMAT Statement

- General form of a FORMAT statement

```
sl    FORMAT (CCC, specifier1, specifier2,...)
```

where

sl = statement label (integer up to 5 digits)

CCC = carriage control character (only for output)

= ' ' single vertical spacing

= '0' double vertical spacing

= '1' new page

= '+' no advance; reset to start of current line

specifier = instruction for individual variable

Edit Descriptors

- Provide detailed formatting information on how data are to be read (for input), printed or displayed (for output). Formatted input is *RARELY* used. Will only concentrate on output from now!

- Two main categories of format specifiers

  1. rules for controlling numerical and character data
  2. rules for controlling physical layout

- Two primary concerns in formatting numerical output

  – total number of spaces
  – total number of significant digits to be displayed

Edit Descriptors: Controlling Numerical and Character Data

Category	Descriptor	Function	Form	Example
Numerical data	I	Integer	Iw	I5
	F	Real	Fw.d	F6.2
	E	Real (exponential)	Ew.d	E12.3
	D	Double Precision	Dw.d	D20.8
	G	Real (general) switches between F and E format	Gw.d	G8.3
Character data	A	Character variable	Aw	A20
	' '	Character strings	'xxx'	'Example'

Edit Descriptors: Controlling Numerical and Character Data

- General form of numerical format descriptor

  TYPE*width*(.*decimals*)

  where

  TYPE = a letter (I, F, E, D or G) indicating the type of data

  width = total width of space desired

  decimals = total number of decimal places (not need for integer)

  Examples:

  `F12.3, E12.5, D20.8`

Edit Descriptors:
Controlling Numerical and Character Data

- Demonstrate **Example 3.5** for `I` edit descriptor.
- Demonstrate **Example 3.6** for `F` edit descriptor.
- Demonstrate **Example 3.8** for `E` edit descriptor.
- Demonstrate **Example 3.8** for `D` edit descriptor.
- Demonstrate **Example 3.11** for `' '` edit descriptor.

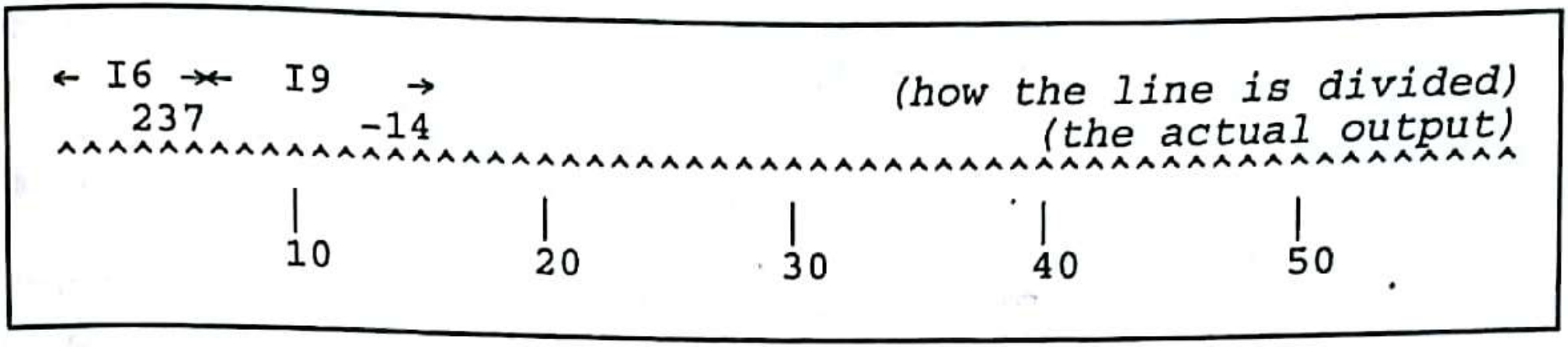
EXAMPLE 3.4

When we print out integers, our only concern is that we leave enough space in the printed line for all the digits of the number. We do not need to worry about the number of decimal places since integers can only be whole numbers. Therefore, the form of the specifier becomes:

I *w*

where I indicates an integer number and *w* indicates the total amount of reserved space. To illustrate, consider the following example:

```
        ICOUNT = 237
        JCOUNT = -14
        PRINT 33, ICOUNT, JCOUNT
33      FORMAT(' ', I6, I9)
```



EXAMPLE 3.5

Consider the following program segment where the program does not allow sufficient space to print out the data:

```
        ICOUNT = 12345
        JCOUNT = -98765
        PRINT 98, ICOUNT, JCOUNT
98      FORMAT(' ', I5, I5)
```

This will result in the following output:

```
← I5→← I5→                                    (set up of the line)
12345*****                                   (the actual printout)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
          |         |         |         |         |
          10        20        30        40        50
```

EXAMPLE 3.6

Here is an example of how to use the F edit descriptor:

```
        DIST  = 12.345
        TIME  = 0.00345
        VELOC = DIST/TIME
        PRINT 5, DIST, TIME, VELOC
5       FORMAT(' ', F7.2, F9.6, F10.1)
```

will produce the following output:

```
← F7.2→← F9.6 →← F10.1 →                    (set up of the line)
  12.35 0.003450    3578.3                  (the actual printout)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
          |         |         |         |         |
          10        20        30        40        50
```

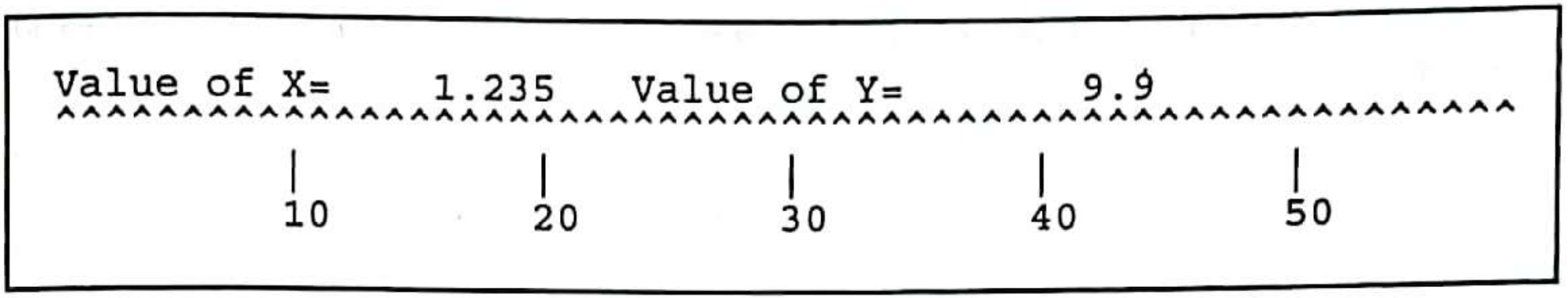
EXAMPLE 3.7

The following example demonstrates how to control the number of decimal digits printed and the spacing between two numbers. Also, we have included strings inside the FORMAT statement.

```
      X = 1.234567
      Y = 9.876543
      PRINT 10, X, Y
10    FORMAT(' ','Value of X= ',F8.3,3X,'Value of Y= ',F9.1)
```

To better understand the format specifiers, we should interpret each one separately:

' '	The carriage control character – start a new line in column 1;
'Value...'	Character string – just print out what is inside the apostrophes;
F8.3	Descriptor for controlling the printout of the first variable, X. F8.3 specifies a total of eight columns with three decimal places;
3X	Skip three spaces;
'Value...'	Another string – do as above;
F9.1	Descriptor for printing the value of the second variable Y in nine columns and one decimal place.


Value of X= 1.235 Value of Y= 9.9
10
20
30
40
50

Scientific Notation

We can also express real numbers in scientific notation as we discussed in Chapter 2. The general form for printing a real number in this exponential format is:

E *w.d*

where E = indicates exponential format (mantissa $\times 10^n$).
w = total width of field reserved for number.
d = desired number of decimal places for mantissa.

As with the F format, the E format has a special rule about how many additional spaces must be reserved. Besides the number of significant digits of the mantissa, the E format requires a total of 7 additional spaces. Thus, the rule for the E*w.d* format is:

$$w \geq d + 7$$

```
      E12.4   →
_ 0.1235E+02
```

7 Required reserved spaces

Double Precision

D *w.d*

where D = indicates double precision format (e.g. 0.123D+003).
w = total width of field reserved for number.
d = desired number of decimal places for mantissa.

The principal difference between E and D formats is that the exponent for double precision can be significantly larger than that for single precision. Therefore, you must allow for a three-digit exponent with the D format compared to two digits for E format. The following rule summarizes these requirements:

$$w \geq d + 8$$

EXAMPLE 3.9

To demonstrate the double precision format, let's reexamine Example 3.8, but now with the D format:

```
      DOUBLE PRECISION DIST, TIME, VELOC
      DIST  = 12.345
      TIME  = 0.00345
      VELOC = DIST/TIME
      PRINT 5, DIST, TIME, VELOC
5     FORMAT(' ', D12.4, D11.3, D10.1)
```

will produce the following output:

```
←   D12.4   →←  D11.3  →←  D10.1 →          (set up of line)
 0.1235D+002 0.345D-002  0.4D+004       (the actual output)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
           |           |           |           |           |
          10          20          30          40          50
```

EXAMPLE 3.9

To demonstrate the double precision format, let's reexamine Example 3.8, but now with the D format:

```
      DOUBLE PRECISION DIST, TIME, VELOC
      DIST  = 12.345
      TIME  = 0.00345
      VELOC = DIST/TIME
      PRINT 5, DIST, TIME, VELOC
5     FORMAT(' ', D12.4, D11.3, D10.1)
```

will produce the following output:

```
←   D12.4   →←  D11.3  →←  D10.1 →          (set up of line)
 0.1235D+002 0.345D-002  0.4D+004       (the actual output)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
           |           |           |           |           |
          10          20          30          40          50
```

```
      X = 12.34
      Y = -0.025
      PRINT 34, X, Y, X*Y
34    FORMAT(' ','X = ',F6.2,' Y = ',F6.3, ' PROD = ',F10.5)
```

will produce the following output:

```
X = ←F6.2→ Y = ←F6.3→ PROD = ←   F10.5 →
X =  12.34 Y = -0.025 PROD =    -0.30850
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
         |         |         |         |         |
        10        20        30        40        50
```

' '	Carriage Control Character – begin new line
'X = '	Character string – Print X =
F6.2	Floating point format – print out first number as XXX.XX
' Y = '	Character string – Print Y =
F6.3	Floating point format – Print out second number as XX.XXX
' PROD = '	Character string – Print PROD =
F10.5	Floating point format – Print out third number as XXXX.XXXXX

Edit Descriptors: Controlling Physical Layout

Category	Descriptor	Function	Form	Example
Spacing	`X`	Individual space	rX	`5X`
	`T`	Tab to column c	Tc	`T20`
	`TR`	Tab right s spaces	TRs	`TR3`
	`TL`	Tab left s spaces	TLs	`TL5`
	`/`	New line	/	/
Repeat	`r( )`	Reuse specifiers	`r( )`	`2(F6.2,I5)`

Descriptor	General Form	Example	Function
X	*n*X	3X	Skip *n* spaces (3 spaces in the example)
/	/	/	Skip to next line
T	T*n*	T32	Tab to column *n* (32 in this example)

EXAMPLE 3.15

Here are a few additional examples to show how the repeat descriptor works:

Original Format	Equivalent Format
F7.3, F7.3, F7.3	3F7.3
/ , / , / (*skip two lines*)	3(/) or ///
F7.3, I6, /, F7.3, I6, /	2(F7.3, I6, /)
F7.3, I6, 2X, I6, 2x, F7.3, I6, 2X, I6	2(F7.3, 2(I6, 2X))
F7.3, 2X, I6, F7.3, 2X, I6, F9.4, I4, F9.4, I4	2(F7.3, 2X, I6), 2(F9.4, I4)

Edit Descriptors:
Controlling Physical Layout

- Use your Fortran compiler to test **Example 3.13**.
- Use your Fortran compiler to test **Example 3.14**.
- Use your Fortran compiler to test **Example 3.15**.

EXAMPLE 3.14

The spacing edit descriptors are easy to use and are very effective in improving the appearance of your output. Below is an example combining all three. Assume that BASE = 12.4, HEIGHT = 9.6 and VOL = 119.04.

```
      PRINT 9, BASE, HEIGHT, VOL
9     FORMAT(' ',5X,F9.3,/,3X,' X',T7,F9.3,/,
     1       T6,'_______',/,T7,F9.3)
```

will produce the following output:

```
~~~~~~~~~12.400
~~~X~~~~~9.600
~~~~~_________
~~~~~~~~119.040
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
          |         |         |         |         |
         10        20        30        40        50
```

Homeworks

- Go through all **Solved Problems** on pages 88–94.

- Use your Fortran compiler to test **Supplementary Problems** 3.15, 3.17, 3.19 and 3.24.
  Hand in your **printed codes** before the end of next meeting.