CHE 208

FORMATTING

Lecture-6

Lubna Ahmed

Input/output

Input/output

- List directed input/output
- Formatted (or user formatted) input/output

List directed Input/output Statements

- List directed I/O are said to be in **free format.**
- Free format is specified by the second asterisk in the READ(*,*) and WRITE(*,*) statements.

Example:

```
WRITE(*,*) "The  output values are", var1,var2,var3
```

- However , the results of writing out data in free format are not always pretty. A large number of extra spaces often appear in the output.
- Hence we shall learn how to write out data using **Formats** that specify the exact way in which the numbers should be printed.

Formatted Input/output Statements

- Format Statement is a non-executable statement and used in conjunction with formatted I/O statements.
- Fortran formats are used to control the appearance of the input and output. It has the following simple form:

(..... format edit descriptors .....)

- That is, a Fortran format is a pair of parenthesis that contains format edit descriptors separated by commas.

Possible Ways to Prepare a Fortran Format

- Write the format as a character string and use it to replace the second asterisk in **READ(*,*) or WRITE(*,*).**

READ(*,'(2I5,F10.2)') ... variables ...

READ(*,"(5F10.2)") ... variables ...

WRITE(*,'(A,I5)') ... variable and expressions ...

WRITE(*,"(10F5.2)") ... variable and expressions ...

Possible Ways to Prepare a Fortran Format cont'd

- Since a format is a character string, we can declare a character constant to hold a format string.

```
CHARACTER(LEN=20), PARAMETER :: FMT1 = "(I5,F10.2)"
CHARACTER(LEN=*), PARAMETER :: FMT2 = "(4I5,E14.7)"
READ(*,FMT1) ... variables ...
READ(*,FMT1) ... variables ...
WRITE(*,FMT2) ... variables and expressions ...
WRITE(*,FMT2) ... variables and expressions ...
```

Possible Ways to Prepare a Fortran Format cont'd

We can also use a character variable to hold a format. In the example below, the character variable String is set to a format and used in READ and WRITE statements.

```
CHARACTER(LEN=80) :: String
String = "(3I5, 10F8.2)"
READ(*,String) ... variables ...
WRITE(*,String) ... variables and expressions ...
```

Format Edit Descriptors

- The tedious part of using Fortran format is to master many format edit descriptors.
- The number of positions to be used is the most important information in an edit descriptor.

Fortran's many Format descriptors fall into four basic categories:

- Format descriptors that describe the vertical position of a line in text
- Format descriptor that describe the horizontal position of data in a line
- Format descriptors that describe the output format of a particular value
- Format descriptors that control the repetition of portions of a FORMAT Statement.

Format Edit Descriptors cont'd

Symbols used with Format descriptors

Symbol	Meaning
c	Column number
d	the number of digits to the right of the decimal point
e	the number of digits in the exponent part
n	Number of spaces to skip
m	the minimum number of positions to be used
w	**Field width:** the number of positions to be used
r	**Repeat Count:** the number of times to use a descriptor or a group of descriptors

Format Edit Descriptors cont'd

<table>
<tr><th colspan="2">Purpose</th><th colspan="2">Edit Descriptors</th></tr>
<tr><td colspan="2">Reading/writing INTEGERs</td><td>Iw</td><td>Iw.m</td></tr>
<tr><td rowspan="4">Reading/writing REALs</td><td>Decimal form</td><td colspan="2">Fw.d</td></tr>
<tr><td>Exponential form</td><td>Ew.d</td><td>Ew.dEe</td></tr>
<tr><td>Scientific form</td><td>ESw.d</td><td>ESw.dEe</td></tr>
<tr><td>Engineering form</td><td>ENw.d</td><td>ENw.dEe</td></tr>
<tr><td colspan="2">Reading/writing LOGICALs</td><td colspan="2">Lw</td></tr>
<tr><td colspan="2">Reading/writing CHARACTERs</td><td>A</td><td>Aw</td></tr>
<tr><td rowspan="3">Positioning</td><td>Horizontal</td><td colspan="2">nX</td></tr>
<tr><td>Tabbing</td><td>Tc</td><td>TLc and TRc</td></tr>
<tr><td>Vertical</td><td colspan="2">/</td></tr>
<tr><td rowspan="3">Others</td><td>Grouping</td><td colspan="2">r(....)</td></tr>
<tr><td>Format Scanning Control</td><td colspan="2">:</td></tr>
<tr><td>Sign Control</td><td colspan="2">S, SP and SS</td></tr>
</table>

INTEGER Output: The I Descriptor

The Iw and Iw.m descriptors are for INTEGER output.

rIw and rIw.m

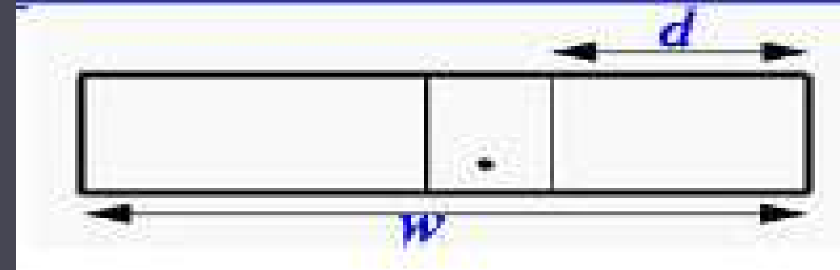
- I is for INTEGER
- w is the width of field, which indicates that an integer should be printed with w positions.
- m indicates that at least m positions (of the w positions) must contain digits. If the number to be printed has fewer than m digits, leading 0s are filled. If the number has more than m digits, m is ignored and in this case Iw.m is equivalent to Iw.

INTEGER Output: The I Descriptor cont'd

- r is the repetition indicator, which gives the number of times the edit descriptor should be repeated. For example, 3I5.3 is equivalent to I5.3, I5.3, I5.3.
- The sign of a number also needs one position. Thus, if -234 is printed, w must be larger than or equal to 4. The sign of a positive number is not printed.
- What if the number of positions is less than the number of digits plus the sign? For example, what if a value of 12345 is printed with I3? Three positions are not enough to print the value of five digits. In this case, all w positions are filled with *'s. Therefore, if you see a sequence of asterisks, you know your edit descriptor does not have enough length to print a number

REAL Output: The F Descriptor

rFw.d



- F is for REAL
- w is the width of field, which indicates that a real number should be printed with w positions.
- d indicates the number of digits after the decimal point.
- The fractional part may have more than d digits. In this case, the (d+1)th digit will be rounded to the dth one
- The fractional part may have fewer than d digits. In this case, trailing zeros will be added.

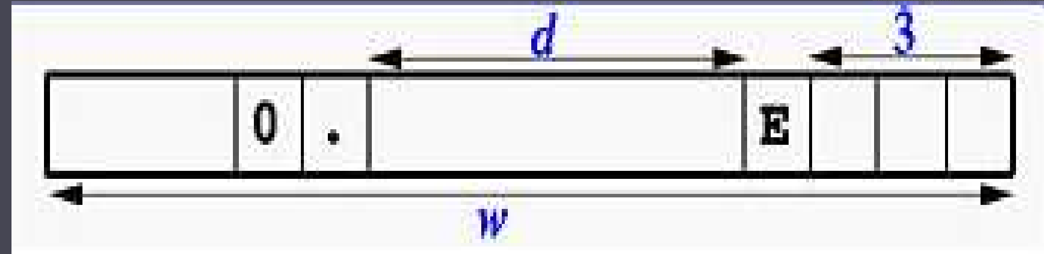
REAL Output: The F Descriptor cont'd

REAL :: a = 123.345, b = -123.345

	Statement	1	2	3	4	5	6	7	8	9	10
1	WRITE(*,"(F10.0)") a							1	2	3	.
2	WRITE(*,"(F10.1)") a						1	2	3	.	3
3	WRITE(*,"(F10.2)") a					1	2	3	.	3	5
4	WRITE(*,"(F10.3)") a				1	2	3	.	3	4	5
5	WRITE(*,"(F10.4)") a			1	2	3	.	3	4	5	0
6	WRITE(*,"(F10.5)") a		1	2	3	.	3	4	5	0	0
7	WRITE(*,"(F10.6)") a	1	2	3	.	3	4	5	0	0	0
8	WRITE(*,"(F10.7)") a	*	*	*	*	*	*	*	*	*	*
9	WRITE(*,"(F10.4)") b		-	1	2	3	.	3	4	5	0
10	WRITE(*,"(F10.5)") b	-	1	2	3	.	3	4	5	0	0
11	WRITE(*,"(F10.6)") b	*	*	*	*	*	*	*	*	*	*

REAL Output: The E Descriptor

rEw.d



- E is for REAL numbers in exponential forms.
- w is the width of field, which indicates that a real number should be printed with w positions.
- To print a number in an exponential form, it is first converted to a normalized form s0.xxx...xxxx×10sxx, where s is the sign of the number and the exponent and x is a digit.

  For example, 12.345, -12.345, 0.00123 -0.00123 are converted to 0.12345×10^{2}, -0.12345×10^{2}, 0.123×10^{-2} and -0.123×10^{-2} respectively

REAL Output: The E Descriptor cont'd

rEw.dEe

- If your data has an exponent larger than 99 or less than -99, Ew.d will not be able to print it properly because there are only two positions for the exponent (therefore, all w positions will be filled with asterisks).
- w must be greater than or equal to d+7

As shown in the figure, in addition to the d positions for the normalized number and e positions for the exponent, we need four more positions for printing the sign in the exponent, a decimal point, a leading 0 and the character E. Moreover, if the number is negative, a sign before the 0 is needed. This means that w must be greater than or equal to d+e+5.

REAL Output: The E Descriptor cont'd

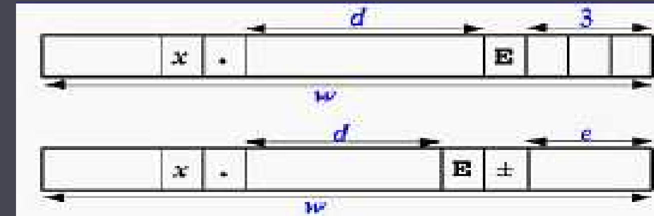
Example: In the following table, the WRITE statements use different E edit descriptors to print the value of 3.1415926.

```
REAL :: PI = 3.1415926
```

	Statement	1	2	3	4	5	6	7	8	9	10	11	12
1	WRITE(*,"(E12.5)") PI		0	.	3	1	4	1	6	E	+	0	1
2	WRITE(*,"(E12.3E4)") PI		0	.	3	1	4	E	+	0	0	0	1
3	WRITE(*,"(E12.7E1)") PI	0	.	3	1	4	1	5	9	3	E	+	1

Editor Descriptor ESw.d and ESw.dEe

ESw.d and ESw.dEe

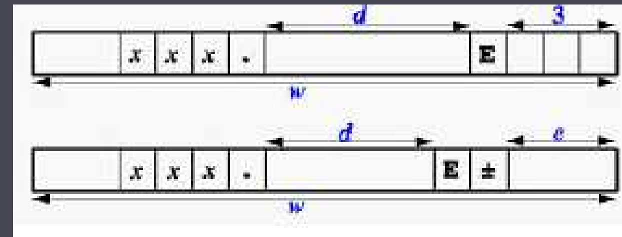


- Scientists write the exponential form in a slightly different way. This ES edit descriptor is for printing a real number in scientific form, which has a non-zero digit as the integral part. If the number is a zero, then all digits printed will be zero.
- For example, if the number is 34.5678, it has a normalized form 0.345678×102. Now shifting the decimal point to the right one position gives 3.45678×101. The following shows the output printed with ES12.3E3:

		3	.	4	5	7	E	+	0	0	1

Editor Descriptor ENw.d and ENw.dEe

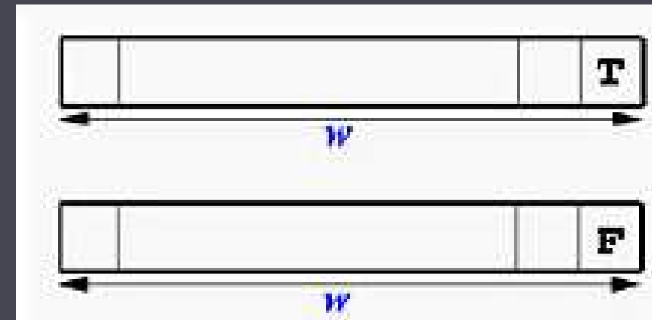
ENw.d and ENw.dEe



- Engineers write the exponential form in yet another way. In an engineering form, the exponent is always a multiple of three, and the printed number always has no more than three and at least one non-zero digits.
- For example, suppose the given number is 1234.567. The integral part has four digits and the exponent is zero. To convert this number to an engineering form, the decimal point should be shifted to the left three positions. Thus, the given number has a new form **1.234567×10^3.**

LOGICAL Output: The L Descriptor

rLw



- L is for LOGICAL
- w is the width of field, which indicates that a logical value should be printed with w positions.
- The output of a LOGICAL value is either T for .TRUE. or F for .FALSE. The single character value is shown in the right-most position and the remaining w-1 positions are filled with spaces. The is shown in the figure below.

```
LOGICAL :: a = .TRUE., b = .FALSE.
WRITE(*,"(L1,L2)") a, b
WRITE(*,"(L3,L4)") a, b
```

Line	1	2	3	4	5	6	7
1	T		F				
2			T				F

CHARACTER Output: The A Descriptor

rAw

- A is for CHARACTER
- w is the width of field
- The output of the character string depends on the length of the character string and the value of w.
- If w is larger than the length of the character string, all characters of the string can be printed and are right-justified. Also, leading spaces will be added. **Example:**

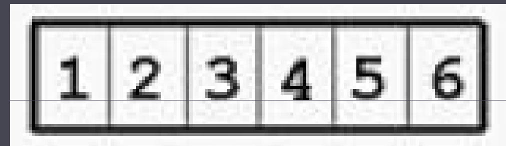
WRITE(*,'(A6)') "12345"

	1	2	3	4	5

CHARACTER Output: The A Descriptor

- If w is less than the length of the character string, then the string is truncated and only the left-most w positions are printed in the w positions. Example:

  WRITE(*,'(A6)') "12345678"



- If w is missing (i.e., edit descriptor A), then the value of w is assumed to be the length of the string.

CHARACTER Output: The A Descriptor

CHARACTER(LEN=5) :: a = "12345"
CHARACTER :: b = "*”

a = "12345" b = "*"

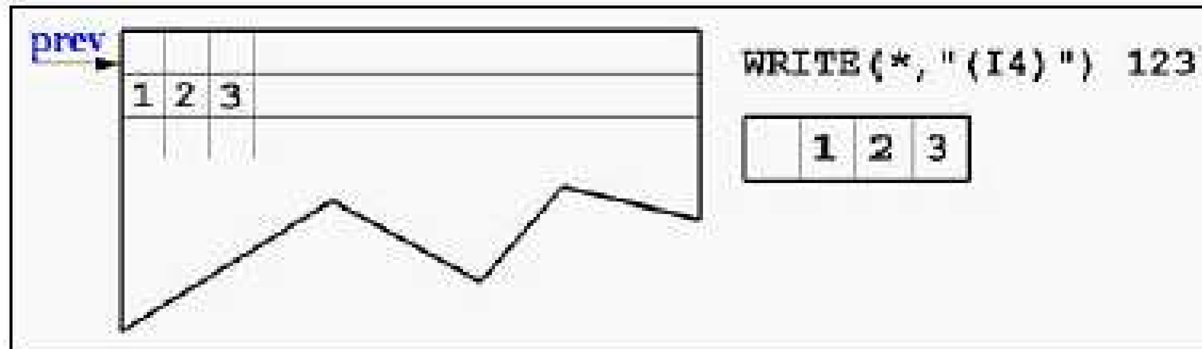
	Statement	Variables	1	2	3	4	5	6	7	8
1	WRITE(*,"(A1,A)")	a, b	1	*						
2	WRITE(*,"(A2,A)")	a, b	1	2	*					
3	WRITE(*,"(A3,A)")	a, b	1	2	3	*				
4	WRITE(*,"(A4,A)")	a, b	1	2	3	4	*			
5	WRITE(*,"(A5,A)")	a, b	1	2	3	4	5	*		
6	WRITE(*,"(A6,A)")	a, b		1	2	3	4	5	*	
7	WRITE(*,"(A7,A)")	a, b			1	2	3	4	5	*
8	WRITE(*,"(A,A)")	a, b	1	2	3	4	5	*		

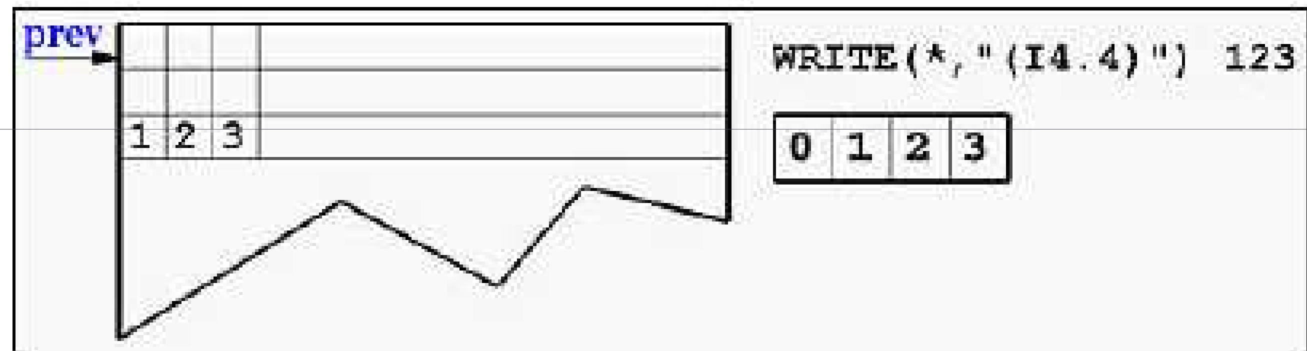
Printer control

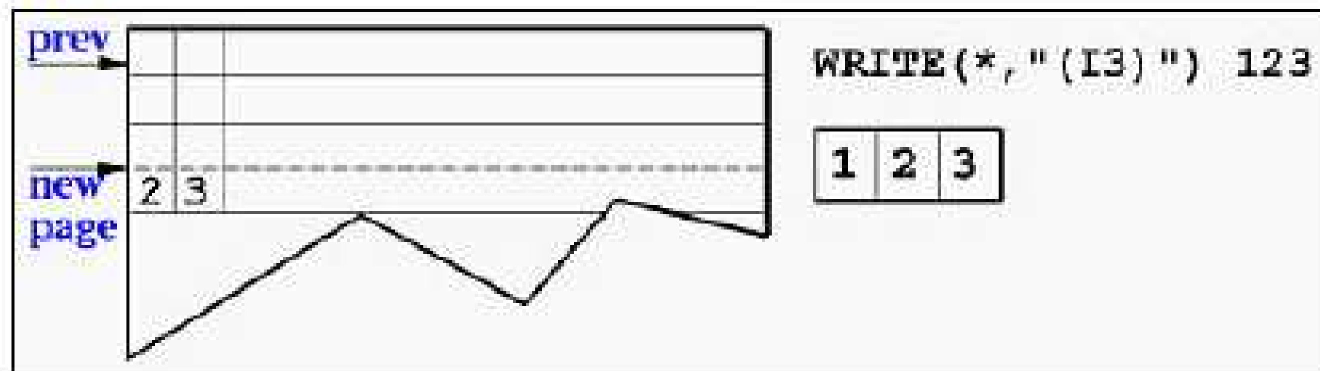
- The control character is not printed in the page. Instead, it provides vertical positioning control information to the printer.
- The way Fortran prints your information is line-oriented

Printer Control Character	Action
1	Skip to new page
Blank	Single spacing
0	Double spacing
+	No spacing(print over previous line)

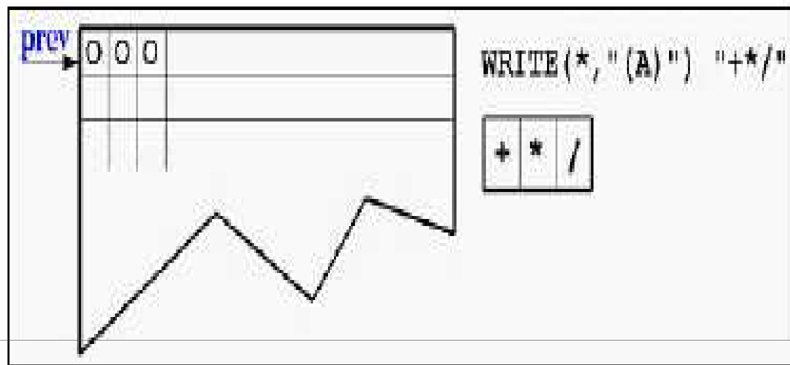
Printer control cont'd

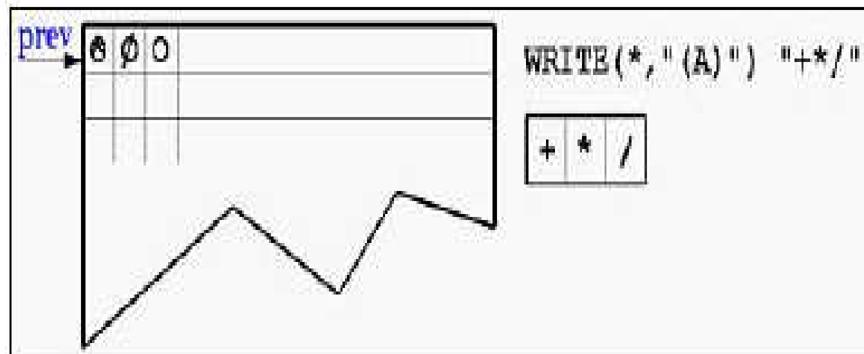






Printer control cont'd





If the first character is a +, then the printer will not advance and print the information on the same line. Therefore, the information on this line will print over the information on the previously printed line.

Horizontal Spacing

nX

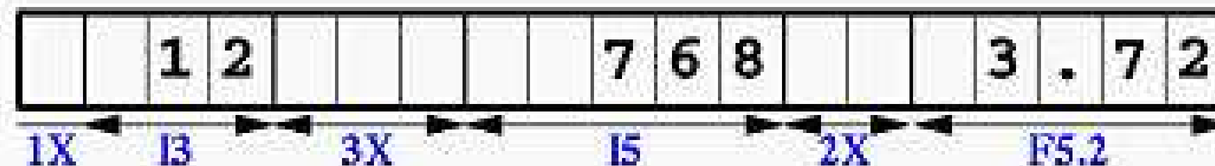
- The next n positions are skipped.
- The X edit descriptor can be used for both input and output. For output, the next n positions are skipped and the content there is unchanged. For input, it simply skips the n positions.
- Unlike edit descriptors I, F, E, L and A, the number of positions is placed before the edit descriptor. Do not write Xn.
- Edit descriptor X cannot be used with repetition indicator .
- The X edit descriptor is only for skipping positions and does not read and write any values.

Horizontal Spacing cont'd

Example:

```
PROGRAM HORIZONTAL
CHARACTER(LEN=30) :: FMT = "(1X,I3,3X,I5,2X,F5.2)"
INTEGER :: a = 12
INTEGER :: b = 768
REAL :: c = 3.723
WRITE(*,FMT) a, b, c
END PROGRAM HORIZONTAL
```

This should generate an output as follows:



Tabbing

c, TLc and TRc

- Tc moves to position c, TLc move backward c positions, and TRc moves forward c positions.
- Edit descriptors T, TL (Left Tab field) and TR(Right Tab field) cannot be used with repetition indicator directly.
- The T, TL and TR edit descriptors are only for tabbing and do not read and write any values.

Tabbing cont'd

Example:

```
PROGRAM Tabbing
INTEGER :: a = 123, b = 456
WRITE(*,"(T6,I4,T2,I4)") a, b
END PROGRAM Tabbing
```

It should generate output as follows:

	4	5	6		1	2	3

Vertical Spacing Control

/ and r/

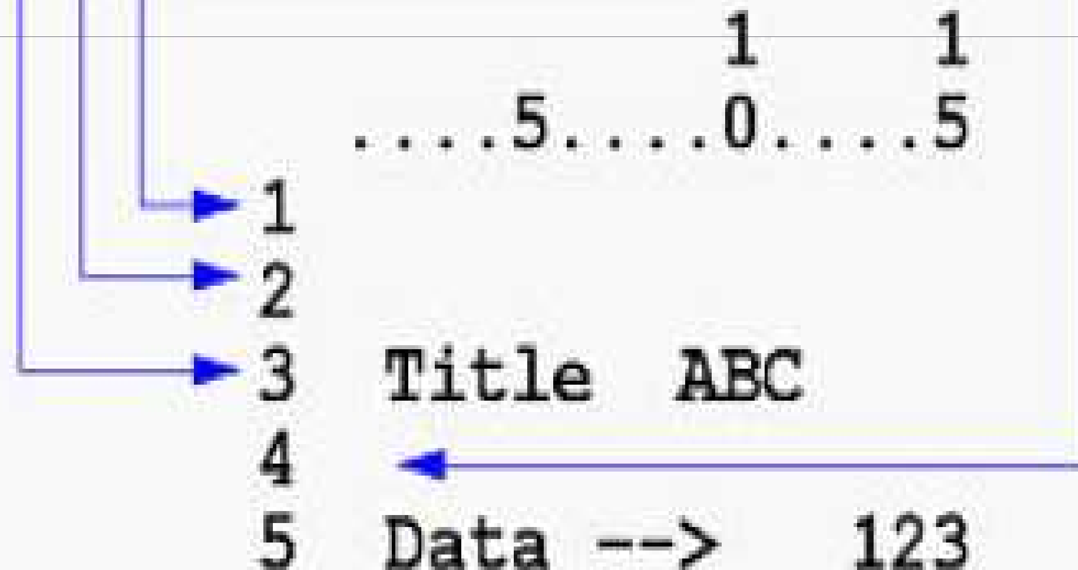
- For Input: The current input line is skipped and the remaining unread content on the current input line is ignored. The reading process starts at the first position on the next input line.
- For Output: The current output line is printed and the next output item starts at the first position of a new output line. READ(*,"(I3,5X,I5,/,/,T15,F10.0)") ......

  is equivalent to

  READ(*,"(I3,5X,I5//T15,F10.0)") ......
- r is equivalent to writing / r times.

Vertical Spacing Control cont'd

```
WRITE(*,"(//1X,A,A/)")  "Title", "  ABC"
WRITE(*,"(1X,A,I5)")    "Data --> ", 123
```



Grouping

r () and ()

- For the form of r(....), the edit descriptors within () repeat r times.
- For example:

  (1X, 3(I5, F5.2), A)

is equivalent to the following:

  (1X, I5, F5.2, I5, F5.2, I5, F5.2, A)

Sign Control

S, SP and SS

- S: All subsequent numeric output is up to your compiler system.
- SP: All subsequent positive numeric output will have plus signs.
- SS: All subsequent positive numeric output will not have plus signs. S, SP, SS can be used multiple times in the same format.
- The effect of S, SP and SS will be applied to all subsequent printing in the same format.

(I5, SP, I6, F6.2 / S, F5.0, SS, I4)

Sign Control cont'd

Example: Write a program which prints the values from -5 to 5. For each value, it is converted to a REAL. Each value will be printed with SP and SS to illustrate the result.

```
PROGRAM SignControl
IMPLICIT NONE
INTEGER :: i
REAL :: x
CHARACTER(LEN=*), PARAMETER :: Format = "(1X,SS,I5,SP,I5,SS,F6.1,SP,F6.1)"
CHARACTER(LEN=*), PARAMETER :: Heading = " SS SP SS SP"
WRITE(*,"(1X,A)") Heading
DO i = -5, 5
x = REAL(i)
WRITE(*,Format) i, i, x, x
END DO
END PROGRAM SignControl
```

End of Lecture…

Class Assignment

- Write a program that for each INTEGER in the range of 1 and 10, prints its value, square, cube, square root and the fourth root. You should generate the output as shown below.

```
 1     1     1   1.0000000   1.0000000
 2     4     8   1.4142135   1.1892071
 3     9    27   1.7320508   1.3160740
                :
                :
                :
10   100  1000   3.1622777   1.7782794
```

Class Assignment

- Rabbit Breeding: the Fibonacci Way

1. Start with one new-born male/female pair.
2. A new-born pair produce a male/female pair after two months.
3. Male/female pairs of age two months and older produce a male/female pair every month. If we represent the number of male/female pairs after n months by the variable Fn, it soon reveals that Fn takes the following values:

Month n1	2	3	4	5	6	7	8
Population F_n	1	1	2	3	5	8	13

Write a program that
Produces this output:
Hint: Fn = Fn_1 + Fn_2

```
Month      Population      Ratio
-----      ----------      -----

  3            2.0         2.0000
  4            3.0         1.5000
...
```