FORMAT Edit Descriptors

A `FORMAT` statement is a labelled nonexecutable statement which can appear anywhere within a program unit. It is of the form

label `FORMAT(`*edit-descriptor-list*`)`

Data descriptors are used to read and write items in the data transfer list in `READ` and `WRITE` statements. Format control descriptors do not correspond to any item in the data transfer list but control other aspects of I/O such as tabulation, new lines, treatment of blanks, etc. Commas must separate most descriptors in the list but can be omitted around some of them like the record control descriptor.

In the following examples, leading, embedded and trailing blanks are denoted as ␣. The leading space `' '` at the beginning of each `FORMAT` statement used for output is a carriage control descriptor, not a printable space character. (Alternatively, you can use `1X` in place of `' '` as the carriage control descriptor.)

Data Descriptors

Symbols

In the following discussion, the letters *w*, *m*, *d* and *e* are unsigned integers which mean the following:

w	total field width (must be greater than zero)
m	minimum number of digits produced on output (may be zero)
d	number of digits to the right of the decimal point (may be zero)
e	number of digits in the exponent part (must be greater than zero)

Any data descriptor can be preceded by a *repeat-count*. The repeat-count is another unsigned integer which indicates how often the descriptor is to be repeated.

Note that the number of positions used in a format descriptor is **not** the precision of that number.

INTEGER Data

`INTEGER` data may not contains decimal points, exponential notation, or any other punctuation (such as commas). The field width should be large enough to include one character for a leading plus or minus sign.

Descriptor	`I`*w*
Input	The `INTEGER` is right-justified in a field *w* characters wide with leading blanks if necessary.
Output	The `INTEGER` is right-justified in a field *w* characters wide with leading blanks if necessary.

Descriptor	`I`*m*
Input	The `INTEGER` is right-justified in a field *w* characters wide with leading blanks if necessary.
Output	The `INTEGER` is right-justified in a field *w* but requires that at least *m* digits are written out, putting in leading zeros if necessary.

Example

```
      I = -1024
      J = 666
      K = 112358
      WRITE(*,100)I,J,K
```

The FORMAT statement

```
  100 FORMAT(' ',I10,I10,I10)
```

begins with the carriage control descriptor `' '` which is followed by three `I10` descriptors. These tell the computer to output three right-justified `INTEGER`s, each with a field width of 10. The numbers are padded with blanks ␣ from the left to make up the required width. The output is

```
␣␣␣␣␣-1024␣␣␣␣␣␣␣666␣␣␣␣112358
```

Note that `I10,I10,I10` is equivalent to `3I10` where 3 is the repeat-count. There is no comma between the repeat-count and its associated descriptor(s).

A slightly different FORMAT statement is

```
  100 FORMAT(' ',3I10.8)
```

which again yields three right-justified `INTEGER`s of width 10 but this time requires a minimum of 8 digits to be printed. Zeros are padded onto the left to make up the required number of digits. The output is

```
␣-00001024␣␣00000666␣␣00112358
```

What happens if the `I` descriptor is too short to accomodate the number (plus any leading sign)?

```
  100 FORMAT(' ',3I5)
```

In this instance three right-justified `INTEGER`s of width 5 are output but since one of the data values is 6 digits long, 5 asterisks * are printed in its place:

```
-1024␣␣666*****
```

REAL, DOUBLE PRECISION and COMPLEX Data

Floating point data (`REAL`, `DOUBLE PRECISION` and `COMPLEX`) can be transferred using `D`, `E`, `F` or `G` descriptors. On output the numbers are rounded to the specified number of digits.

The F notation produces a decimal or fixed-point notation which is particularly useful in formatting tables but cannot be used for very large or very small numbers. The field width must be large enough to include one character for a leading plus or minus sign in addition to a decimal point. In practice, this means $w > d + 1$.

Descriptor	F*wd*
Input	A decimal point always takes precedence. If there is a decimal point, then *d* is ignored and the number is read into the *w* positions. If there is no decimal point, then the last *d* digits are taken to be the decimal part of the number and the *w-d* digits before those are taken to be the whole part of the number.
Output	The floating point number is right-justified in a field *w* characters wide. This means *d* decimal places, a decimal point and *w-d-1* places preceding the decimal point which must include any leading plus or minus sign.

Example

```
      A = 3.14159
      B = -88.9
      C = 123.4567E-02
      WRITE(*,200)A,B,C
```

The FORMAT statement

```
  200 FORMAT(' ',F7.2,F7.2,F7.2)
```

begins with the carriage control descriptor `' '` which is followed by three `F6.2` descriptors. These tell the computer to output three right-justified floating point numbers, each with a field width of 7 and with 2 digits after the decimal point. The output appears as

```
␣␣␣3.14␣-88.90␣␣␣1.23
```

Values are rounded as necessary to fit the width and digit specifications. Even though C is given explicitly in exponential notation, it can be written with the F descriptor. Again, note that `F7.2,F7.2,F7.2` could have been written more simply with a repeat-count as `3F7.2`.

If we alter the FORMAT statement to

```
  200 FORMAT(' ',3F7.4)
```

to include more places after the decimal point, what happens?

```
␣3.1416*******␣1.2346
```

Recall that the field width must include the decimal point and any leading sign. The value `-88.9` requires a field width that is at least 4 spaces longer than the number of digits following the decimal point. Since the space alloted is too small, 7 asterisks * are printed instead.

The E descriptor produces exponential notation which is useful for very large and very small numbers. The field width *w* must accomodate

- a optional leading plus or minus sign,
- an optional digit in front of the decimal point,
- the decimal point,
- the required number of digits *d* after the decimal point,
- the D or E exponent designator,
- the exponent sign (plus or minus) and
- the number of digits *e* in the exponent.

As a result, *w* has to be quite a bit bigger than *d*. In practice, this means *w* > *d* + 4 + *e* where *e* is 2 unless explicitly stated otherwise.

Note that the D notation is identical to the E notation on input. On output it produces a D instead of an E exponent designator. It is used for `DOUBLE PRECISION` numbers.

Descriptor	D*wd* , E*wd*
Input	Identical to F*w.d* descriptor.
Output	The floating point number is written in exponential notation with a mantissa of *d* digits and is right-justified in a field *w* characters wide.

Descriptor	D*wd* E*e* , E*in d* E*e*
Input	Identical to F*w.d* descriptor.
Output	The floating point number is written in exponential notation with an exponent of *e* digits, a mantissa of *d* digits and is right-justified in a field *w* characters wide.

This second form explicitly states how many digits are in the exponent.

Example

```
      DOUBLE PRECISION D
      A = 3.14159
      B = -88.9
      C = 123.4567E-02
      D = 2.99792458D+08
      WRITE(*,201)A,B,C,D
```

The FORMAT statement

```
  201 FORMAT(' ',3E10.2,D10.2)
```

begins with the carriage control descriptor `' '` which is followed by four descriptors, three `E10.2` (using a repeat-count) and one `D10.2`. These tell the computer to output four right-justified floating point numbers in exponential format, each with a field width of 10 and with 2 digits after the decimal point. The fourth number is printed with a D exponent designator rather than the usual E exponent designator. The output looks like

```
␣␣0.31E+01␣-0.89E+02␣␣0.12E+01␣␣0.30D+09
```

Another example shows all of the numbers being printed with more digits. It is permissible to output DOUBLE PRECISION numbers using the E data descriptor just as it is permissible to output REAL numbers using the E data descriptor.

```
  201 FORMAT(' ',4E15.7)
```

requires a field width of 15 and 7 digits after the decimal point for 4 right-justified floating point numbers written in exponential notation. The result is

```
␣␣0.3141590E+01␣-0.8890000E+02␣␣0.1234567E+01␣␣0.2997925E+09
```

As in the case of the F descriptor, values are rounded as necessary in order to achieve the required number of digits after the decimal point.

The field width must be quite a bit larger than the number of digits after the decimal point in order to allow for a leading sign, a leading digit, a decimal point and at least four spaces for the exponent field. If the field width is too small, the output will be asterisks `*`.

The G descriptor can be used if you don't know ahead of time the magnitude of your numbers.

Descriptor	G*wd*
Input	Identical to F*w.d* descriptor.
Output	If the value is greater than 0.1 but not too large to fit in the field, then the number is written as if in decimal format (similar to the F descriptor) using a field width of *w* and outputting *d* significant digits. This number is followed by four blanks. Otherwise, it behaves exactly like the E*w.d* descriptor.

Descriptor	G*wd* E*email*
Input	Identical to F*w.d* descriptor.
Output	This form allows you to specify the length of exponent to be *e* digits. As before, if the value is greater than 0.1 but not too large to fit in the field, then the number is written as if in decimal format using a field width of *w* and outputting *d* significant digits. This number is followed by *e* + 2 blanks. Otherwise, it behaves exactly like the E*w.d* descriptor.

Example

```
      DOUBLE PRECISION D
      A = 3.14159
      B = -88.9
      C = 123.4567E-02
      D = 2.99792458D+08
      WRITE(*,202)A,B,C,D
  202 FORMAT(' ',4G10.2)
```

The `FORMAT` statement begins with the carriage control descriptor `' '` which is followed by four `G10.2` descriptors (using a repeat-count). These tell the computer to output four floating point numbers in either fixed point or exponential format, depending on the magnitude of the number. The result is

```
␣␣␣3.1␣␣␣␣␣␣-89.␣␣␣␣␣␣␣1.2␣␣␣␣␣␣0.30e + 09
```

The first three numbers are larger than 0.1 but not 'too large' so they are printed with 2 significant digits and followed by 4 blanks ␣. (They are padded out on the left with blanks ␣ to achieve the required field width of 10.) The fourth number is 'large' so it is printed out as if an `E10.2` descriptor had been used.

Note that a `COMPLEX` number requires two data descriptors, one for the real part and one for the imaginary part. They may `D`, `E`, `F`, `G` or any combination, as long as there are two of them.

Example

```
      COMPLEX T
      T = (-15.8,309.67)
      WRITE(*,203)T
  203 FORMAT(' ',2F10.5)
```

The `FORMAT` statement begins with the carriage control descriptor `' '` which is followed by two `F10.5` descriptors (using a repeat-count). These tell the computer to output 2 right-justified floating point numbers in decimal format, each with a field width of 10 and with 5 digits after the decimal point. The output is

```
␣-15.80000␣309.67001
```

Why does the number `309.67` appear as `309.67001` when written out to five decimal places? Floating point numbers that cannot be expressed as combinations of powers of 2 will not be exactly representable when stored internally in binary. The computer will store the number as the nearest binary value, either single-precision or double-precision. Then, when the value is output, it is translated back into decimal but it may not be exactly the decimal value you started with. Thus, a number like 0.1 may appear as 0.1000000 or 0.1000001 or even 0.9999999, depending on the compiler. There is also the problem of precision. The computer can only store so many digits for each number and if you attempt to print out more, you end up with whatever garbage happens to be occupying that memory space. Do not mistake the number of digits you print out with the actual precision of the number.

CHARACTER Data

If no field width is specified, then the length of the `CHARACTER` item determines it.

Descriptor	A or A*w*
Input	If the length *k* of the `CHARACTER` variable is shorter than the field width *w*, then the rightmost *k* characters are used. If the length *k* is greater than *w*, then *w* characters will be read in, with the trailing *k*-*w* positions of the `CHARACTER` variable consisting of blanks.
Output	If the length *k* of the `CHARACTER` data is shorter than the field width *w*, then the value is right-justified and preceded with *w*-*k* blanks. If the length *k* is greater

than *w*, then the first (leftmost) *w* characters of CHARACTER data is written and the rest ignored.

Example

```
      CHARACTER P*11,Q*20
      P = 'Hello world'
      Q = 'The cow jumped'
      WRITE(*,300)P,Q
```

First, it is important to understand how values are stored in the CHARACTER variables. The content of P exactly matches the declared length of the variable but the content of Q is shorter than the declared length so the content is padded out on the right with blanks ␣.

```
P : Hello␣world
Q : The␣cow␣jumped␣␣␣␣␣␣
```

The FORMAT statement

```
  300 FORMAT(' ',A11,A20)
```

begins with the carriage control descriptor `' '` which is followed by two A descriptors, the first of width 11 and the second of width 20. These widths exactly match the declared sizes of the variables and the output is

```
Hello␣worldThe␣cow␣jumped␣␣␣␣␣␣
```

The exact same effect can be obtained by using `A,A` in place of `A11,A20`. The A descriptor with no declared field width automatically takes the length of the corresponding item in the data transfer list.

If the width *w* is shorter than the length of the value to be printed, then only the first *w* characters are used and the rest are discarded. However, if the width *w* is longer than the length of the value to be printed, then the value is right-justified and blanks ␣ are added to the left to pad it out to the required length. In

```
  300 FORMAT ( '', A5, A25)
```

the first 5 positions in the first variable P are printed, followed by Q. However, because Q is 20 characters long but has been given 25 spaces to fill, 5 blanks ␣ are prepended to it. The output looks like this:

```
Hello␣␣␣␣␣The␣cow␣jumped␣␣␣␣␣␣
```

LOGICAL Data

Descriptor	L*w*
Input	The input field must contain the letter T or the letter F within the *w* characters specified. The letters T or F may be preceded by a period and any number of blanks. Anything appearing after the letters T or F is ignored.
Output	The output field is *w* characters wide and consists of the letter T or the letter F preceded by *w-1* blanks.

Example

```
      LOGICAL X,Y
      X = .FALSE.
      Y = .TRUE.
      WRITE(*,400)X,Y
  400 FORMAT(' ','X =',L2,' and Y =',L5)
```

The FORMAT statement begins with the carriage control descriptor `' '` which is followed by a string `'The value of X is'`, a logical descriptor of width 2, another string `' and the value of Y is'`, and another logical descriptor, this time of width 5. The output from these statements is

```
X␣=␣F␣and␣Y␣=␣␣␣␣T
```

The string is written out, followed by an F for `.FALSE.`, right-justified in a field width of 2. Another string follows, then a T for `.TRUE.`, right-justified in a field width of 5. Strings or character constants are written out exactly as they appear.

Format Control Descriptors

Symbols

In the following discussion, the letter *k* is a signed integer and the letter *n* is an unsigned integer.

Character Constants

A character constant or string (text appearing within apostrophes) may appear in FORMAT statements associated with output only. The character constant is simply output as is.

Another method is to use a Hollerith string. Named in honour of Herman Hollerith, an American inventer who founded a company that later became part of IBM, the H descriptor has the form *n*H*string* where *string* is exactly *n* characters long.

The Hollerith string is a deprecated feature in FORTRAN 77 and its use is strongly discouraged.

Example

```
      CHARACTER Q*20
      Q = 'The cow jumped'
      WRITE(*,500)Q
```

The FORMAT statement

```
  500 FORMAT(' ',A14,' over the moon.')
```

begins with the carriage control descriptor `' '` which is followed by an A descriptor (14 characters wide) and a character constant or string `' over the moon.'`. These tell the computer to print the first 14 characters of the variable, followed by the string exactly as it is given. The output is

```
The cow jumped over the moon.
```

The equivalent FORMAT statement using a Hollerith string instead of a character constant is

```
  500 FORMAT(' ',A14,15H over the moon.)
```

The descriptor 15H tells the compiler that the next 15 characters are a character constant and should be output exactly as is. The use of character constants is far less prone to error than Hollerith strings.

Blank Control

The default value depends on the BLANK= item in the OPEN statement and is restored at the start of every new formatted transfer.

Descriptor	***Action***
BN	All embedded and trailing blanks are treated as null and ignored. This descriptor applies only to input and is ignored on output.
BZ	All embedded and trailing blanks are treated as zeros. This descriptor applies only to input and is ignored on output.

Example

Suppose that you have an input file with the values `1␣-23␣456␣-7890` all in one record (line). The following code is used to read it in:

```
      OPEN(1,FILE='input.dat',STATUS='OLD')
      READ(1,600)I,J,K
  600 FORMAT(I1,I3,I6)
      CLOSE(1)
```

The data descriptors don't exactly match the way the numbers are arranged in the data file so the numbers stored in the variables I, J and K may not be what you expect or want.

- `I = 1`
- `J = -2`
- `K = 3456`

What happened? The first number in the file and the first data descriptor are both INTEGERs of length 1 so I takes that value. However, the second number in the file is -23, yet J is assigned the value -2. That's because the next 3 positions in the file are ␣-2. Finally, the third number K takes the next 6 characters in the file which are 3␣456␣. The blanks ␣ are ignored so the number becomes 3456.

The results get even stranger if you use the BZ descriptor in the FORMAT statement. Then all blanks ␣ are regarded as zeros.

```
      OPEN(1,FILE='input.dat',STATUS='OLD')
      READ(1,600)I,J,K
  600 FORMAT(BZ,I1,I3,I6)
      CLOSE(1)
```

The values stored in the variables now are

- `I = 1`
- `J = -2`
- `K = 304560`

When using formatted input, you must know exactly how the file is arranged. Now see what happens if you instead use an unformatted READ statment:

```
      OPEN(1,FILE='input.dat',STATUS='OLD')
      READ(1,*)I,J,K
      CLOSE(1)
```

The values stored in the variables are what you probably want:

- `I = 1`
- `J = -23`
- `K = 456`

Plus Sign Control

In all instances the minus - sign is always printed.

Descriptor	***Action***
S	Restore the system default for printing plus + signs. This is normally SS.
SP	Force the printing of plus + signs in front of numerical data for the remainder of the FORMAT description. This descriptor applies only to output and is ignored on input.
SS	Suppress the printing of plus + signs in front of numerical data for the remainder of the FORMAT description. This descriptor applies only to output and is ignored on input.

Example

```
      A = 3.14159
      B = -88.9
      C = 123.4567E-02
      I = -1024
      J = 666
      K = 112358
      WRITE(*,601)I,A,J,B,K,C
  601 FORMAT(' ',SP,I6,F6.2,I8,F8.2,SS,I12,E12.4)
```

The FORMAT statement begins with the carriage control descriptor `' '` which is immediately followed by the SP descriptor, activating the printing of plus + signs. Then follows an I descriptor and a F descriptor, both of width 6, and another pair of I and F descriptors, these of width 8. The SS descriptor then suppresses the printing of plus + signs for the last two numbers which are formatted by an I descriptor and an E descriptor, both of width 12. The output is

```
␣ -1024 ␣ +3.14 ␣ ␣ ␣ ␣ +666 ␣ ␣ -88.90 ␣ ␣ ␣ ␣ ␣ ␣ 112 358 ␣ ␣ 0.1235E + 01
```

Scaling Factor

The scaling factor is used to scale floating point values by a factor of 10^k where *k* is a (small) signed integer. The default value for *k* is 0.

Descriptor	*k*P
Input	If there is an explicit exponent in the input value, the scaling factor has no effect. Otherwise, the number is multiplied by 10^{-k} before input, thus changing the value.
Output	If there is an explicit exponent in the output value, the mantissa is multiplied by 10^k and the exponent is reduced by *k*, thus leaving the value unchanged. Otherwise, the number is multiplied by 10^k before output, thus changing the value.

Example

```
      A = 3.14159
      B = -88.9
      C = 123.4567E-02
      WRITE(*,602)A,B,C
  602 FORMAT(' ',F10.4,F10.4,E12.5)
      WRITE(*,603)A,B,C
  603 FORMAT(' ',-2P,F10.4,F10.4,E12.5)
```

Both FORMAT statements begin with the carriage control descriptor `' '` and contain two F descriptors and one E descriptor. However, in the second FORMAT statement, a scaling factor

-2P of 10^{-2} is applied before the numbers are printed. The output appears as

```
␣ ␣ ␣ ␣ 3.1416 ␣ ␣ -88.9000 ␣ 0.12346E + 01
 ␣ ␣ ␣ ␣ 0.0314 ␣ ␣ ␣ -0.8890 ␣ 0.00123E + 03
```

When a scaling factor is used with an F descriptor, the value is actually changed upon input or output. However, the value is unchanged when there is an explicit exponent in the number. In this case, the mantissa is shifted and the exponent changed to keep the number the same.

Column Position Control

These descriptors control which column position you are in a record.

Descriptor	***Action***
*n*X	Shift right *n* spaces from current position. Note that the number *n* is placed before the descriptor X, not after.
T*n*	Shift to column *n* in the record.
TL*n*	Shift left *n* spaces from current position. Note that you cannot shift left past column 1. This descriptor allows you to reread records on input and overwrite records on output.
TR*n*	Shift right *n* spaces from current position. Note that TR*n* is exactly equivalent to *n*X.

Example

```
      CHARACTER P*11,Q*20
      P = 'Hello world'
      Q = 'The cow jumped'
      WRITE(*,700)Q(5:7),P
```

The FORMAT statement

```
  700 FORMAT(' ',TR6,A,'. How are you?',TL23,A5)
```

begins with the carriage control descriptor `' '` as usual. Then tab right 6 spaces before printing a CHARACTER variable. The A descriptor is used so the field width is exactly the same as the corresponding value in the data transfer list. In this case, the variable to be output is `Q(5:7)`, a 3-letter substring of Q. This is followed by a character constant `'. How are you?'`. At this point, you tab left 23 places and write out a 5-letter CHARACTER variable. The output is

```
Hello␣cow.␣How␣are␣you?
```

The FORMAT statement

```
  700 FORMAT(' ',6X,A,' How are you?',T1,A5)
```

accomplishes exactly the same thing. The 6X descriptor gives 6 spaces before the first A descriptor and character constant. Then T1 instructs the computer to tab to column 1 before acting on the last A descriptor. The X descriptor for spacing is possibly one of the most useful descriptors available.

Record Control

The slash / descriptor begins a new line (record) on output and skips to the next line on input, ignoring any unread information on the current record. Two slashes // skips one line, three slashes /// skips two lines, etc.

Example

```
      CHARACTER P*11,Q*20
      S = 1024.0
      P = 'Hello world'
      Q = 'The cow jumped'
      WRITE(*,701)P,Q(5:7),'Did you know that the square root of',
     $            S,SQRT(S)
  701 FORMAT ( '', A5,1X, A, /, '', A, 1X, F6.1, '=', F6.1, '?', //,
     $       ' ',20X,'It is!')
```

The data transfer list contains five element, a CHARACTER variable of length 11, a CHARACTER substring of length 3, a character constant, and two REAL numbers. (SQRT(S) is a call to the intrinsic square root function SQRT and it returns a REAL value.) As always the FORMAT statement begins with the carriage control descriptor ' '. Then it prints out the first five characters stored in P, followed by a single space and then the substring Q(5:7). The record control descriptor / then tells the computer to start a new line. This is followed by a carriage control descriptor. On this new line the A descriptor controls the output for the character string. A single space follows, then a floating point number of width 6. The character constant ' = ' is next and then another floating point number, followed by yet another character constant '?'. The double record control descriptor // starts a new line and then another new line immediately, effectively double-spacing. The final line of output is 20 spaces followed by the string 'It is!'. Here's the final product:

```
Hello␣cow
Did␣you␣know␣that␣the␣square␣root␣of␣1024.0␣=␣␣␣32.0?

␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣It is!
```

Forced Reversion and Scan Control

If there are fewer items in the data transfer list than there are data descriptors, then all of the unused descriptors are simply ignored. However, if there are more items in the data transfer list than there are data descriptors, then **forced reversion** occurs. In this case, FORTRAN 77 advances to the next record and rescans the format, starting with the right-most left parenthesis, including any repeat-count indicators. It then re-uses this part of the format. If there are no inner parenthesis in the FORMAT statement, then the entire format is reused.

Forced reversion has no effect on plus sign control, blank control or scaling factors.

The colon descriptor : is used to terminate format scanning immediately if there are no more items in the data transfer list.

Example

What happens if there is a mismatch in the number of data transfer list items and data descriptors?

```
      A = 3.14159
      B = -88.9
      C = 123.4567E-02
      I = -1024
      J = 666
      K = 112358
      WRITE(*,702)I,J,K
  702 FORMAT(' ',6I10)
      WRITE(*,*)
      WRITE(*,703)I,A,J,B,K,C
  703 FORMAT ( '', SP, I10, F10.4)
```

The output is

```
␣␣␣␣␣-1024␣␣␣␣␣␣␣666␣␣␣␣112358

␣␣␣␣␣-1024␣␣␣+3.1416
␣␣␣␣␣␣+666␣␣-88.9000
␣␣␣+112358␣␣␣+1.2346
```

In the first WRITE/FORMAT pair, there are three INTEGERs to be printed but the FORMAT statement has 6 I descriptors. Thus, the three values are printed out according to the FORMAT statement and the rest of the unused descriptors are ignored.

The unformatted WRITE(*,*) statement simply outputs a blank line.

In the second WRITE/FORMAT pair, there are six numbers (three sets of INTEGER/REAL pairs) to be printed but there are only two data descriptors. (Recall that the SP descriptor turns on plus + sign printing.) After the first two numbers are output, forced reversion occurs. Since there are no internal parentheses in the FORMAT statement, the entire statement is reused. In this example, it is reused twice more before all of the values are written out. Every time the FORMAT statement is reused, in whole or in part, a new record (line) is started. Thus, the output from the second WRITE/FORMAT pair covers three lines.

Example

```
      INTEGER          NMAX
      PARAMETER(NMAX=6)
      DOUBLE PRECISION PHYSIX(NMAX)
      DATA PHYSIX /6.02214129D+23,1.3806488D-23,96485.3365,
     $ 483597.870D + 09,6.62606957D-34,10973731.568539 /
      WRITE(*,704)PHYSIX(1),PHYSIX(2),PHYSIX(3),PHYSIX(4),
     $ PHYSIX (5) PHYSIX (6)
  704 FORMAT(' ',E14.7,2X,:,'There is more to come!')
```

This is another case of forced reversion when there are more items in the data transfer list than data descriptors. Because of the scan control descriptor : in the FORMAT statement, once all the items in the data transfer list are used, anything after the : is ignored. This is why the string 'There is more to come!' is not printed the last time.

```
+ 24-0.6022141E ␣ ␣ There ␣ IS ␣ More ␣ to ␣ Come!
22-0.1380649E ␣ ␣ There ␣ IS ␣ More ␣ to ␣ Come!
+ 05 0.9648534E ␣ ␣ There ␣ IS ␣ More ␣ to ␣ Come!
0.4835979E + 15 ␣ ␣ Top All ␣ Is ␣ More ␣ To ␣ COME!
33-0.6626070E ␣ ␣ There ␣ IS ␣ More ␣ to ␣ Come!
+ 08 0.1097373E ␣ ␣
```

If the scan control descriptor : is placed before the second 2X instead of after, then the final line of output will simply be

```
0.1097373E + 08
```

In other words, the two final spaces will not be included in the output.

Carriage Control

These four characters (also known as ASA carriage control characters) control the pagination.

Character	***Vertical Spacing Before Printing***
`' '`	advance one line
`'0'`	advance two lines
`'1'`	advance one page

`'+'`	do not advance

These control characters only work if the output device recognises them. Unfortunately, not all systems do so the results can be unpredictable. Normal practice is to put a blank or 1X at the start of each FORMAT statement and after each slash /. If following this convention, special care must be taken in cases of forced reversion to make sure a blank is the first item in each new record.

More Examples

In the following examples, leading, embedded and trailing blanks are denoted as ␣.

Example

It is permissible to have a READ or WRITE statement with no data transfer list at all. For example, the program fragment

```
      WRITE(*,800)
  800 FORMAT(' ',50('-'))
```

writes out 50 dashes -, effectively making a dashed line 50 characters long. The output looks like this:

```
--------------------------------------------------
```

Example

An implied DO loop is often used with arrays to write out data in a neat tabular form.

```
      INTEGER          I,NMAX
      PARAMETER(NMAX=6)
      CHARACTER*30     LABEL(NMAX)
      DOUBLE PRECISION PHYSIX(NMAX)
      LABEL(1)  = 'Avogadro constant'
      LABEL(2)  = 'Boltzmann constant'
      LABEL(3)  = 'Faraday constant'
      LABEL(4)  = 'Josephson constant'
      LABEL(5)  = 'Planck constant'
      LABEL(6)  = 'Rydberg constant'
      PHYSIX (1) + 23 = 6.02214129D
      PHYSIX (2) = 1.3806488D-23
      PHYSIX (3) = 96485.3365
      PHYSIX (4) + 09 = 483597.870D
      PHYSIX (5) = 6.62606957D-34
      PHYSIX (6) = 10973731.568539
      WRITE(*,801)(I,LABEL(I),PHYSIX(I),I=1,NMAX)
  801 FORMAT(' ',I2,3X,A,1X,1P,E14.7)
```

The output consists of three columns. In each line there is an INTEGER of width 2, 3 spaces, a CHARACTER (of length 30 in this instance), another space, and finally a floating point number written in exponential form (width 14 and 7 digits after the decimal). The floating point number is scaled by a factor of 10^1. This is what it looks like:

```
␣ . 1 ␣ ␣ ␣ Avogadro ␣ Constant ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ 6.0221413E + 23 is
 ␣ 2 ␣ ␣ ␣ BOLTZMANN ␣ Constant ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ 1.3806488 23 is-E
 ␣ . 3 ␣ ␣ ␣ Faraday ␣ Constant ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ 9.6485336E + 04
 ␣ . 4 ␣ ␣ ␣ JOSEPHSON ␣ Constant ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ 4.8359787E + 14
 ␣ . 5 ␣ ␣ ␣ Planck ␣ Constant ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ 6.6260696E-34 is
 ␣ . 6 ␣ ␣ ␣ of Rydberg ␣ Constant ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ 1.0973732E + 07
```

Because the implied DO loop in the WRITE statement has 18 items in the data transfer list and the FORMAT statement has only three data descriptors in it, forced reversion occurs five times, leading to a nicely formatted table over six lines.

Example

Finally, forced reversion can lead to unexpected results when the FORMAT statement contains interior parentheses.

```
      INTEGER          I,NMAX
      PARAMETER(NMAX=6)
      DOUBLE PRECISION D,PHYSIX(NMAX)
      DATA PHYSIX /6.02214129D+23,1.3806488D-23,96485.3365,
     $ 483597.870D + 09,6.62606957D-34,10973731.568539 /
      D = 2.99792458D+08
      WRITE(*,802)D,(PHYSIX(I),I=1,NMAX)
```

The output for the FORMAT statement

```
  802 FORMAT(' ',2X,D15.7,2(2X,E15.7))
```

appears as follows:

```
␣␣␣␣ 0.2997925D + 09 ␣␣␣␣ 0.6022141E + 24 ␣␣␣␣ 0.1380649E-22 is
 ␣␣␣␣ 0.9648534E + 05 ␣␣␣␣ 0.4835979E + 15
 ␣␣␣␣ 0.6626070E-33 is ␣␣␣␣ 0.1097373E + 08
```

The WRITE statement contains a total of seven numbers to print out but there are only three data descriptors in the FORMAT statement so forced reversion occurs. The first pass through the FORMAT statement yields 2 spaces, a number written in double-precision exponential format, and then the parenthetical group containing 2 spaces and a number written in single-precision exponential format which has a repeat-count of 2 in front of it. When forced reversion occurs, only the 2(2X,E15.7) part of the FORMAT statement is reused.

Compare this with the FORMAT statement

```
  802 FORMAT(' ',2X,D15.7,2X,E15.7,2X,E15.7)
```

Although 2(2X,E15.7) is equivalent to 2X,E15.7,2X,E15.7, they are treated differently when forced reversion occurs. The output for the FORMAT statement without the internal parentheses is

```
␣␣␣␣ 0.2997925D + 09 ␣␣␣␣ 0.6022141E + 24 ␣␣␣␣ 0.1380649E-22 is
 ␣␣␣␣ 0.9648534D + 05 ␣␣␣␣ 0.4835979E + 15 ␣␣␣␣ 0.6626070E-33 is
 ␣␣␣␣ 0.1097373D + 08
```

In the second case, the entire FORMAT statement is reused, not just a subset of it.