Chapter 9

Format Specification

This chapter contains the following subsections:

- "FORMAT Statement"
- "Field and Edit Descriptors"
- "Field Descriptor Reference"
- "Edit Descriptor Reference"
- "Complex Data Editing"
- "Interaction Between I/O List and Format"
- "List−Directed Formatting"

A format specification provides explicit editing information to the processor on the structure of a formatted data record. It is used with formatted I/O statements to allow conversion and data editing under program control. An asterisk (*) used as a format identifier in an I/O statement specifies list−directed formatting.

A format specification may be defined in a **FORMAT** statement or through the use of arrays, variables, or expressions of type character. During input, field descriptors specify the external data fields and establish correspondence between a data field and an input list item. During output, field descriptors are used to describe how internal data is to be recorded on an external medium and to define a correspondence between an output list item and an external data field.

This section describes the **FORMAT** statement, field descriptors, edit descriptors, and list−directed formatting. It also contains a discussion of carriage−control characters for vertical control in printing formatted records.

As extensions to Fortran 77, the compiler supports additional processor−dependent capabilities, which are described in the *Fortran 77 Programmer's Guide*.

Format specifications can be given in two ways: in **FORMAT** statements or as values of character arrays, character variables, and other character expressions.

Format Stored as a Character Entity

In a formatted input or output statement, the format identifier can be a character entity, provided its value has the syntax of a format specification, as detailed below, on execution. This capability allows a character format specification to be read in during program execution.

When the format identifier is a character array name, the format specification is a concatenation of all the elements in the array. When the format identifier is a character array element name, the format specification is only that element of the array. Therefore, format specifications read through a character array name can fill the whole array, while those read through a character array element name must fit in a single element of that array.

FORMAT Statement

The **FORMAT** statement is a non–executable statement that defines a format specification. It has the following syntax:

xx FORMAT *fs*

where

xx	is a statement number that is used as an identifier in a **READ**, **WRITE**, **PRINT**, or **ASSIGN**(label) statement.
fs	is a format specification (described in ''Format Specification'').

Format Specification

The syntax of a format specification fs is

([*flist*])

where *flist* is a list of format specifiers of one of the following forms, separated by commas:

[*r*]*fd*
ed
[*r*]*fs*

where

r	is a positive integer specifying the repeat count for the field descriptor or group of field descriptors. If *r* is omitted, the repeat count is assumed to be 1.
fd	is a repeatable edit descriptor or a field descriptor.
ed	is a nonrepeatable edit descriptor.
fs	is a format group and has the same form as a complete format specification except the *flist* must be non–empty (it must contain at least one format specifier).

The comma used to separate the format specifiers in *flist* can be omitted as follows:

- Between a **P** edit descriptor and immediately following an **F**, **E**, **D**, or **G** edit descriptor (see ''P Edit Descriptor'').
- Before or after a slash edit descriptor (see ''Slash Editing'').
- Before or after a colon edit descriptor (see ''Colon Descriptor'').

Descriptors

Some descriptors can be repeated, others cannot. The repeatable descriptors are

I*w[.m]*	**Z***w[.m]*	**E***w.d[***E***e]*	**G***w.d[***E***e]*	**A***[w]*
O*w[.m]*	**F***w.d*	**D***w.d*	**L***w*	**Q**

where

w and *e*	are nonzero, unsigned integer constants.
d and *m*	are unsigned integer constants.

These descriptors are described in the respective section.

The nonrepeatable descriptors are

/	*k***P**	**TR***c*	**SS**	*n***H***h*…	**$**
:	**T***c*	**S**	**BN**	’*h*… ’	
*n***X**	**TL***c*	**SP**	**BZ**	"*h*… "	

where

n and *c*	are nonzero, unsigned integer constants.
k	is an optionally signed integer constant.
h	is one of the characters capable of representation by the processor.

Format Specifier Usage

Each *field descriptor* corresponds to a particular data type I/O list item:

- Integer field descriptors—**I***w*, **I***w.m*, **O***w*, **Z***w*
- Real, double–precision, and complex field descriptors—**F***w.d*, **E***w.d*, **E***w.d***E***e*, **D***w.d*, **G***w. d*, **G***w.d***E***e*
- Logical field descriptor—**L***w*
- Character and Hollerith field descriptors—**A**, **A***w*

O*w*, and **Z***w* are extensions to Fortran 77.

The terms *r, c, n, d, m, e,* and *w* must all be unsigned integer constants, and, additionally, *r, c, n, e*, and *w* must be nonzero. *k* is an optionally signed integer constant. Descriptions of these list items are given in the sections that describe the individual field descriptors.

The repeat specifier *r* can be used only with the **I**, **O**, **Z**, **F**, **E**, **D**, **G**, **L**, and **A** field descriptors and with format groups.

The *d* is required in the **F**, **E**, **D**, and **G** field descriptors. **E***e* is optional in the **E** and **G** field descriptors and invalid in the others.

Use of named constants anywhere in a format specification is not allowed.

Table 9–1contains an alphabetical summary of the field and edit descriptors.

Table 9–1 Summary of Field and Edit Descriptors

Form	Effect
A[*w*]	Transfers character or Hollerith values
BN	Specifies that embedded and trailing blanks in a numeric input field are to be ignored
BZ	Specifies that embedded and trailing blanks in a numeric input field are

	to be treated as zeros
D*w.d*	Transfers real values (D exponent field indicator)
E*w.d*[E*e*]	Transfers real values (E exponent field indicator)
F*w.d*	Transfers real values
G*w.d*	Transfers real values: on input, acts like F descriptor; on output, acts like E or F descriptor, depending on the magnitude of the value
*n*Hc…c	Transfers values between H edit descriptor and an external 'h…' (output only)
I*w*[.*m*]	Transfers decimal integer values
L*w*	Transfers logical values
O*w*[.*m*]	Transfers octal integer values
kP	Scale factor for F, E, D, and G descriptors
S	Restores the default specification for SP and SS
SP	Writes plus characters (+) for positive values in numeric output fields
SS	Suppresses plus characters (+) for positive values in numeric output fields
Tc	Specifies positional tabulation
TLc	Specifies relative tabulation (left)
TRc	Specifies relative tabulation (right)
nX	Specifies that *n* column positions are to be skipped
Z*w*[.*m*]	Transfers hexadecimal integer values
:	Terminates format control if the I/O list is exhausted
/	Record terminator
$	Specifies suppression of line terminator on output (ignored on input)

Each of the field descriptors described Table 9−1is discussed in detail in the following sections.

Variable Format Expressions

Variable format expressions provide a means for substituting run−time expressions for the field width and other parameters of the field and edit descriptors of the statement. Any expression can be enclosed in angle brackets (<>) and used as an integer constant would be used in the same situation. This facility is not available for anything other than a compile−time **FORMAT** statement.

```
Here is an example that uses a variable format expression:
    program VariableExample
    character*12 greeting
    greeting = 'Good Morning!'
    do 110 I = 1, 12
    write (*,115) (greeting)
 115 format (A<I>)
 110 continue
     end
```

In the above example, the field descriptor for greeting has the format **A***w* where *w* is a variable width specifier **I** (initially set to 1) for the *iolist* item **greeting**. In twelve successive **WRITE** operations, **I** is incremented by 1 to produce the following output:

```
G
Go
Goo
Good
Good
Good M
Good Mo
Good Mor
Good Morn
Good Morni
Good Mornin
Good Morning
```

The following rules apply to variable format expressions:

- Functions calls, references to dummy, and any valid Fortran expression can be specified.
- Non−integer data types are converted to integers before processing.
- The same restrictions on size that apply to any other format specifier also apply to the value of a variable format expression.
- Run−time formats cannot use variable format descriptions.
- If the value of a variable changes during a **READ** or **WRITE** operation, the new value is used the next time it is referenced in an I/O operation.

General Rules for Using FORMAT

Because **FORMAT** allows *exact* specification of input and output format, it is necessarily complex. Some guidelines to its correct usage are outlined below.

- A **FORMAT** statement must always be labeled.
- In a field descriptor such as *r***I***w*[.*m*] or *n***X**, the terms *r*, *w*, and *n* must be unsigned integer constants greater than zero. The term *m* must be an unsigned integer constant whose value is greater than or equal to zero; it cannot be a symbolic name of a constant. The repeat count *r* can be omitted.
- In a field descriptor such as **F***w.d*, the term *d* must be an unsigned integer constant. *d* must be specified with **F**, **E**, **D**, and **G** field descriptors, even if *d* is zero. The decimal point is also required. Both *w* and *d* must be specified. In a field descriptor such as **E***w.d***E***e*, the term *e* must also be an unsigned, nonzero integer constant.
- In an **H** edit descriptor such as *n***H***c1 c2.\ .\ .c subn*, exactly *n* characters must follow the **H**. Any character in the processor character set can be used in this edit descriptor.
- In a scale factor of the form *k***P**, *k* must be an optionally signed integer constant. The scale factor affects the **F**, **E**, **D**, and **G** field descriptors only. Once a scale factor is specified, it applies to all subsequent real field descriptors in that format specification until another scale factor appears; *k* must

be zero (**0P**) to reinstate a scale factor of zero. A scale factor of **0P** is initially in effect at the start of execution of each I/O statement.

- No repeat count *r* is permitted in **BN**, **BZ**, **S**, **SS**, **SP**, **H**, **X**, **T**, **TR**, **TL**, **:**, **/**, **$**, **'** descriptors unless these descriptors are enclosed in parentheses and treated as a format group.
- If the associated I/O statement contains an I/O list, the format specification must contain at least one **I**, **O**, **Z**, **F**, **E**, **D**, **G**, **L**, or **A** field descriptor.
- A format specification in a character variable, character substring reference, character array element, character array, or character expression must be constructed in the same way as a format specification in a **FORMAT** statement, including the opening and closing parentheses. Leading blanks are permitted, and any characters following the closing parenthesis are ignored.
- The first character in an output record generally contains carriage control information. See ''Output Rules Summary'' and ''Carriage Control''.
- A slash (**/**) is both a format specifier list separator and a record terminator. See ''Slash Editing'' for details.
- During data transfers, the format specification is scanned from left to right. A repeat count, *r*, in front of a field descriptor or group of field descriptors enclosed in parentheses causes that descriptor or group of descriptors to be repeated *r** before left to right scanning is continued.

Input Rules Summary

Guidelines that apply specifically to input are

- A minus sign (−) must precede a negative value in an external field; a plus sign (+) is optional before a positive value.
- An external field under **I** field descriptor control must be in the form of an optionally signed integer constant, except that leading blanks are ignored and the interpretation of embedded or trailing blanks is determined by a combination of any **BLANK =** specifier and any **BN** or **BZ** blank control that is currently in effect (see ''BN Edit Descriptor'' and ''BZ Edit Descriptor'').
- An external field under **F**, **E**, **D**, or **G** field descriptor control must be in the form of an optionally signed integer constant or a real constant, except that leading blanks are ignored and the interpretation of embedded or trailing blanks is determined by a combination of any **BLANK =** specifier and any **BN** or **BZ** blank control that is currently in effect (see ''BN Edit Descriptor'' and ''BZ Edit Descriptor'').
- If an external field contains a decimal point, the actual size of the fractional part of the field, as indicated by that decimal point, overrides the *d* specification of the corresponding real field descriptor.
- If an external field contains an exponent, the current scale factor *k***P** descriptor has no effect for the conversion of that field.
- The format specification together with the input list must not attempt to read beyond the end of a

record.

Output Rules Summary

Guidelines that apply specifically to output are

- A format specification cannot specify more output characters than the value in the record length specifier (see ‘‘OPEN’’ of Chapter 8 for details). For example, a line printer record might be limited to no more than 133 characters, including the carriage−control character.
- The field−width specification,*w*, and exponent digits, *e*, must be large enough to accommodate all characters that the data transfer can generate, including an algebraic sign, decimal point, and exponent. For example, the field width specification in an **E** field descriptor should be large enough to contain $d + 6$ characters or $d + e + 4$ characters. The first character of a record of a file intended to be printed is typically used for carriage control; it is not printed. The first character of such a record should be a space, 0, 1, or +. (See ‘‘Carriage Control’’.)

Field and Edit Descriptors

The format specifiers in a format specification consist of field, or repeatable, descriptors and other nonrepeatable edit descriptors.

On input, the field descriptors specify what type of data items are to be expected in the external field so that data item values can be properly transferred to their internal (processor) representations.

On output, the field descriptors specify what type of data items should be written to the external field.

On input and output, the other nonrepeatable edit descriptors position the processor pointer in the external field so that data items will be transferred properly. For instance, edit descriptors can specify that lines or positions in the external field be skipped or that data items can be repeatedly read (on input) or written (on output).

Field Descriptor Reference

This section contains an overview of the numeric field descriptors **I**, **O**, **Z**, **F**, **E**, **D**, and **G**. It also describes the **P** edit descriptor and the **L**, **A**, **H**, **Q**, and character edit descriptors.

Numeric Field Descriptors

The **I**, **O**, **Z**, **F**, **E**, **D**, and **G** field descriptors are used for numeric editing. This section also describes the**P** edit descriptor, which is a scale factor, that alters the effect of **F**, **E**, **D**, and **G** field descriptors.

Unless otherwise indicated, the following rules apply:

- On input, these numeric field descriptors ignore leading blanks in the external field. If a**BZ** edit descriptor is in effect, embedded and trailing blanks are treated as zeros; otherwise, a**BN** edit descriptor is in effect, and all embedded and trailing blanks are ignored. Either **BZ** or **BN** is initially in effect at the beginning of the input statement depending on the **BLANK =** specified (see ‘‘OPEN’’). The default is **BN**.

- A plus sign (+) is produced on output only if **SP** is in effect; however, a minus sign (-) is produced where applicable. When computing the field width for numeric descriptors, one character should be allowed for the sign, whether it is produced or not.
- For input with **F**, **E**, **D**, and **G** descriptors, a decimal point in the input field overrides the **D** specification, and an explicit exponent in the input field overrides the current scale factor.
- For output, fields are right justified. If the field width is too small to represent all required characters, asterisks are produced. This includes significant digits, sign, decimal point, and exponent.

Default Field Descriptor Parameters

You can optionally specify a field−width value (*w*, *d*, and *e*) for the **I**, **O**, **Z**, **L**, **F**, **E**, **D**, **G**, and **A** field descriptors. If you do not specify a value, the default values shown in Table 9−2apply. The length of the I/O variable determines the length *n* for the **A** field descriptor.

Table 9−2 Default Field Descriptors

Descriptor	Field Type	w	d	e
I,O,Z	BYTE	7		
I,O,Z	INTEGER*2, LOGICAL*2	7		
I,O,Z	INTEGER*4, LOGICAL*4	12		
O,Z	REAL*4	12		
O,Z	REAL*8	23		
O,Z	REAL*16	44		
L	LOGICAL	2		
F,E,G,D	REAL, COMPLEX*8	15		
F,E,G,D	REAL*8, COMPLEX*16	25	16	2
F,E,G,D	REAL*16	42	33	3
A	LOGICAL*1	1		
A	LOGICAL*2, INTEGER*2	2		
A	LOGICAL*4, INTEGER*4	4		
A	REAL*4, COMPLEX*8	4		
A	REAL*8, COMPLEX*16	8		
A	REAL*26	16		
A	CHARACTER*n	n		

I Field Descriptor

The **I** field descriptor is used for conversion between an internal integer data item and an external decimal integer. It has the form

I*w*[.*m*]

where

w is a nonzero, unsigned integer constant denoting the size of the external field, including blanks and a sign, if necessary. A minus sign (−) is always printed on output

	if the number is negative. If the number is positive, a plus sign (+) is printed only if SP is in effect.
m	is an unsigned integer constant denoting the minimum number of digits required on output. *m* is ignored on input. The value of *m* must not exceed *w*; if *m* is omitted, a value of 1 is assumed.

In an input statement, the **I** field descriptor reads a field of *w* characters from the record, interprets it as an integer constant, and assigns the integer value to the corresponding I/O list item. The corresponding I/O list element must be of the **INTEGER** or **LOGICAL** data type. The external data must have the form of an integer constant; it must not contain a decimal point or exponent.

A **LOGICAL** data type is displayed as either the value 0 (false) or 1 (true).

If the first nonblank character of the external field is a minus sign, the field is treated as a negative value. If the first nonblank character is a plus sign, or if no sign appears in the field, the field is treated as a positive value. An all−blank field is treated as a value of zero.

Table 9−3contains input examples.

Table 9−3I Field Input Examples

Format	External Field	Internal Value
i4	3244	3244
i3	−15	−15
i9	213	213

In an output statement, the **I** field descriptor constructs an integer constant representing the value of the corresponding I/O list item and writes it to the right−justified record in an external field w characters long. If the value does not fill the field, leading blanks are inserted; if the value exceeds the field width, the entire field is filled with asterisks. If the value of the list item is negative, the field will have a minus sign as its left most, nonblank character. The term *w* must therefore be large enough to provide for a minus sign, when necessary. If *m* is present, the external field consists of at least *m* digits, with leading zeros, if necessary.

If *m* is zero, and the internal representation is zero, the external field is filled with blanks.

Table 9−4contains output examples.

Table 9−4I Field Output Examples

Format	Internal Value	External Field
I3	311	311
i4	−311	−311
i5	417	417
i2	7782	**
i3	−213	***
i4.2	1	01
i4.4	1	0001
i4.0	1	

O Field Descriptor

The **O** field descriptor transfers data values and converts them to octal form. It has the form

O*w*[*m*]

where

w is a nonzero, unsigned integer constant denoting the size of the external field, including blanks and a sign, if necessary. A minus sign (−) is always printed on output if the number is negative. If the number is positive, a plus sign (+) is printed only if **SP** is in effect.

m is an unsigned integer constant denoting the minimum number of digits required on output. *m* is ignored on input. The value of *m* must not exceed *w*; if *m* is omitted, a value of 1 is assumed.

This repeatable descriptor interprets and assigns data in the same way as the **I** field descriptor, except that the external field represents an *octal* number constructed with the digits 0 through 7. On input, if **BZ** is in effect, embedded and trailing blanks in the field are treated as zeros; otherwise, blanks are ignored. On output, **S**, **SP**, and **SS** do not apply.

In an input statement, the field is terminated when a non−octal digit is encountered. Fortran 77 treats embedded and trailing blanks as zeros.

In an input statement, the **O** field descriptor reads *w* characters from the record; the input field must have:

- optional leading blanks
- an optional plus or minus sign
- a sequence of octal digits (0 through 7)

A field that is entirely blank is treated as the value zero.

Table 9−5contains examples of **O** field input values. **BN** is assumed in effect, and internal values are expressed in decimal (base 10).

Table 9−5O Field Input Examples

Format	External Field (INTEGER*4)	Internal Value
o20	−77	−63
o20	1234	668
o20	177777	65535
o20	100000	32768

In an output statement, the **O** field descriptor constructs an octal number representing the unsigned value of the corresponding I/O list element as follows:

- The number is right justified with leading zeros inserted (if necessary). Fortran 77 inserts leading blanks.

- If *w* is insufficient to contain all the digits necessary to represent the unsigned value of the output list item, then the entire field is filled with asterisks.

Table 9−6lists examples of **O** field output.

Table 9−6O Field Output Examples

Format	Internal Value (INTEGER*4)	External Field
o20.2	3	03
o20.2	−1	37777777777
o3	−1	***
o20.2	63	77
O20.2	−2	37777777776

Z Field Descriptor

The **Z** field descriptor transfers data values and converts them to hexadecimal form. It has the form

Z*w*[*m*]

where

w is a nonzero, unsigned integer constant denoting the size of the external field.

m is an unsigned integer constant denoting the minimum number of digits required on output. *m* is ignored on input. The value of *m* must not exceed *w*; if *m* is omitted, a value of 1 is assumed.

This repeatable descriptor interprets and assigns data in the same way as the **I** field descriptor, except that the external field represents a hexadecimal number constructed with the digits 0 through 9 and the letters A through F. On output, the output list item is interpreted as an unsigned integer value.

In an input statement, the **O** field descriptor reads *w* characters from the record. After embedded and trailing blanks are converted to zeros or ignored, as applicable, the input field must have

- optional leading blanks
- an optional plus or minus sign
- a sequence of hexadecimal digits (0 through 9, A through F)

A field that is entirely blank is given a value of zero.

Table 9−7lists examples of **Z** field input. **BN** is assumed in effect, and internal values are expressed in decimal (base 10).

Table 9−7Z Field Input Examples

Format	External Field (INTEGER*4)	Internal Value
Z10	−ff	−255
z10	1234	4660
z10	ffff	65535

z10	8000	32768

Table 9−8lists examples of **Z** field output.

Table 9−8 Z Field Output Examples

Format	Internal Value (INTEGER*4)	External Field
z10.2	3	" 03"
z10.2	−1	" ffffffff"
z10.2	63	" 3f"
z10.2	−2	" fffffffe"

F Field Descriptor

The **F** field descriptor transfers real values. It has the form

`F`*w*`.`*d*

where

w is a nonzero, unsigned integer constant denoting field width.

d is an unsigned integer constant denoting the number of digits in the fractional part.

The corresponding I/O list element must be of type **REAL**, **DOUBLEPRECISION**, or **COMPLEX**.

In an input statement, the **F** field descriptor reads a field of *w* characters from the record and, after appropriate editing of leading, trailing, and embedded blanks, interprets it as an integer or a real constant. It then assigns the real value to the corresponding I/O list element. (Refer to Chapter 2, "Constants and Data Structures," for more information.) If the external field contains an exponent, the letter E can be omitted as long as the value of the exponent is a signed integer. If the first nonblank character of the external field is a minus sign, the field is treated as a negative value. If the first nonblank character is a plus sign, or if no sign appears in the field, the field is treated as a positive value. An all−blank field is given a value of zero.

If the field contains neither a decimal point nor an exponent, it is treated as a real number in which the right most *d* digits are to the right of the decimal point, with leading zeros assumed if necessary. If the field contains an explicit decimal point, the location of that decimal point overrides the location specified by the value of *d* in the field descriptor. If the field contains a real exponent, the effect of any associated scale factor *k***P** (see Scale Factor) is suppressed, and the real exponent is used to establish the magnitude of the value in the input field before it is assigned to the list element.

Table 9−9provides examples of **F** field input.

Table 9−9F Field Input Examples

Format	External Field	Internal Value
f8.5	123456789	0.12345678E+03
f8.5	−1234.567	−0.123456E+04
f8.5	12.34e+2	0.1234E+02
F5.2	1234567.89	0.12345E+03

In an output statement, the **F** field descriptor constructs a basic real constant representing the value of the corresponding I/O list element, rounded to *d* decimal positions, and writes it to the record right−justified in an external field *w* characters long.

The term *w* must be large enough to include:

- a minus sign for a negative value or a plus sign (when **SP** is in effect) for a positive value
- the decimal point
- *d* digits to the right of the decimal

If *w* is insufficiently large, the entire field width is filled with asterisks. Therefore, *w* must be > *d* + 2.

Table 9−10provides examples of **F** field output.

Table 9−10F Field Output Examples

Format	Internal Value	External Field
F8.5	.12345678E+01	1.23457
f9.3	.87654321E+04	8765.432
F2.1	.2531E+02	**
f10.4	.1234567E+02	12.3457
f5.2	.123456E+03	******
F5.2	−.4E+00	−0.40

E Field Descriptor

The **E** field descriptor transfers real values in exponential form. It has the form

E*w*.*d*[E*e*]

where

w is a nonzero, unsigned integer constant denoting field width.

d is an unsigned integer constant denoting the number of digits in the fractional part.

e is a nonzero, unsigned integer constant denoting the number of digits in the exponent part. The *e* has no effect on input.

The corresponding I/O list element must be of **REAL**, **DOUBLEPRECISION**, or **COMPLEX** data type.

In an input statement, the **E** field descriptor interprets and assigns data in exactly the same way as the**F** field descriptor.

Table 9−11provides examples of **E** field input.

Table 9−11E Field Output Examples

Format	External Field	Internal Value
e9.3	" 654321E3"	.654321E+06
e12.4	" 1234.56E−6"	.123456E−02
e15.3	"12.3456789"	.123456789E+02

e12.5	"123.4567d+10"	.1234567E+13

In Table 9−11, the**E** field descriptor treats the **D** exponent field indicator the same as an **E** exponent indicator.

In an output statement, the **E** field descriptor constructs a real constant representing the value of the corresponding I/O list element, rounded to *d* decimal digits, and writes it to the right−justified record in an external field *w* characters long. If the value does not fill the field, leading spaces are inserted; if the value exceeds the field width, the entire field is filled with asterisks.

When an **E** field descriptor is used, data output is transferred in a standard form. This form consists of

- minus sign for a negative value or a plus sign (when **SP** is in effect) for a positive value
- digits to the left of the decimal point, if any, or an optional zero
- decimal point
- *d* digits to the right of the decimal point
- an *e* + 2−character exponent or a 4−character exponent

The exponent has one of the following forms:

E*w.d*	**E** + *nn* or **E** -*nn* if the value of the exponent is in the range of -99 to +99
E*w.d*	+*nnn* or -*nnn* if the value of the exponent is <= -99 or <= +99
E*w.d***E***e*	**E** + *n1 n2…n* sub *e* or **E**- *n1 n2…n sub e*, where *n1 n2… ne* is the magnitude of the exponent with leading zeros, if necessary.

The exponent field−width specification is optional; if it is omitted, the exponent part is as shown above. If the exponent value is too large to be output with the given value *e* as shown in the third form above, the entire field is filled with asterisks.

The term *w* must be large enough to include

- A minus sign when necessary (plus signs when **SP** is in effect)
- All significant digits to the left of the decimal point
- A decimal point
- *d* digits to the right of the decimal point
- The exponent

Given these limitations and assuming a **P** edit descriptor is in effect, *w* is \xb3 *d* + 7, or \xb3 *d* + *e* + 5 if *e* is present.

Table 9−12provides examples of **E** field output.

Table 9−12E Field Output Examples

Format	Internal Value	External Field

E9.2	.987654321E+06	" .99E+06"
e12.5	.987654321E+06	" .98765E+06"
e12.3	.69E-5	" .690E-05"
e10.3	-.5555E+00	" –.556E+00"
e5.3	.7214E+02	"*****"
e14.5E4	-.1001E+01	" -.10010E+0001"
e14.3E6	.123e-06	" .123E-000003"

D Field Descriptor

The **D** field descriptor transfers real values in exponential form. It has the form

D*w*.*d*

where

w is a nonzero, unsigned integer constant denoting field width.

d is an unsigned integer constant denoting the number of digits in the fractional part.

The corresponding I/O list element must be of **REAL**, **DOUBLEPRECISION**, or **COMPLEX** data type.

In an input statement, the **D** field descriptor interprets and assigns data in exactly the same way as the**F** field descriptor.

Table 9–13provides examples of **D** field input.

Table 9–13D Field Input Examples

Format	External Field	Internal Value
d10.2	"12345 "	.12345E+03
d10.2	" 123.45"	.12345E+03
d15.3	"123.4567891D+04"	.1234567891E+07

In an output statement, the **D** field descriptor is the same as the **E** field descriptor, except the **D** exponent field indicator replaces the **E** indicator.

Table 9–14provides examples of **D** field output.

Table 9–14D Field Output Examples

Format	Internal Value	External Field
d14.3	123d – 04	" .123D – 04"
d23.12	123456789123d + 04	" .123456789123D + 04"
d9.5	14D + 01	"*********"

G Field Descriptor

A **G** field descriptor is used for the conversion and editing of real data when the magnitude of the data is unknown. On output, the **G** field descriptor produces a field as do the **F** or **E** field descriptors, depending on the value. On input, the **G** field descriptor interprets and assigns data in exactly the same way as the**F** field descriptor. It has the form

`Gw.d[Ee]`

where

w	is a nonzero, unsigned integer constant denoting field width.
d	is an unsigned integer constant denoting the number of digits in the basic value part.
e	is a nonzero, unsigned integer constant denoting the number of digits in the exponent part.

The corresponding I/O list element must be of **REAL**, **DOUBLEPRECISION**, or **COMPLEX** data type.

In an input statement, the **G** field descriptor interprets and assigns data in exactly the same way as the**F** field descriptor.

In an output statement, the **G** field descriptor constructs a real constant representing the value of the corresponding I/O list element rounded to *d* decimal digits and writes it to the right−justified record in an external field *w* characters long. The form in which the value is written is a function of the magnitude of the value *m*, as described in Table 9−1 In the table, *n* is 4 if **E***e* was omitted from the **G** field descriptor; otherwise *n* is *e* + 2.

Table 9−15illustrates the effect of data magnitude on **G** format conventions.

Table 9−15 Effect of Data Magnitude on G Format Conventions

Data Magnitude	Effective Format
m < 0.1	E*w*.*d*[E*e*]
0.1 \xb2 m < 1.0	F(*w*−*n*).*d*, *n* ('")
1.0 \xb2 m < 10.0	F(*w*−*n*).(*d*–1) ('')
10 d−2 \xb2 m < 10d−1	F(*w*−*n*).1,*n* ('')
10 d−1 \xb2 m < 10d	F(*w*−*n*).0*n* ('')
m \xb3 10d	E*w*.*d*[E*e*]

The term *w* must be large enough to include

- A minus sign for a negative value or a plus sign (when **SP** is in effect) for a positive value
- A decimal point
- *d* digits in the basic value part
- Either a 4−character or*e* + 2−character exponent part

Given these limitations, *w* must therefore be \xb3 *d* + 7 or \xb3 *d* + *e* + 5.

Table 9−16provides examples of **G** field output.

Table 9−16G Field Output Examples

Format	Internal Value	External Field
g13.6	.1234567E–01	" .1234567E–01"
g13.6	–.12345678E00	" –.123457 "

g13.6	.123456789E+01	" 1.23457 "
g13.6	.1234567890E+02	" 12.3457 "
g13.6	.12345678901E+03	" 123.457 "
g13.6	−.123456789012E+04	" −1234.57 "
g13.6	.1234567890123E+05	" 12345.7 "
g13.6	.12345678901234E+06	" 123457. "
g13.6	−.123456789012345E+07	" −.123457E+07"

For comparison, the examples in Table 9−17use the same values with an equivalent **F** field descriptor.

Table 9−17 Field Comparison Examples

Format	Internal Value	External Field
f13.6	.1234567E−01	" .012346"
f13.6	−.12345678E00	" −.123457"
f13.6	.123456789E+01	" 1.234568"
f13.6	.1234567890E+02	" 12.345679"
f13.6	.12345678901E+03	" 123.456789"
f13.6	−.123456789012E+04	" −1234.567890"
f13.6	.1234567890123E+05	" 12345.678901"
f13.6	.12345678901234E+06	"123456.789012"
F13.6	−.123456789012345E+07	"*************"

P Edit Descriptor

The **P** edit descriptor specifies a *scale factor* and has the form

*k*P

where *k* is an optionally signed integer constant called the *scale factor.*

A **P** edit descriptor can appear anywhere in a format specification but must precede the first field descriptor that is to be associated with it. For example

*k*PF*w.d*   *k*PE*w.d*   *k*PD *w.d*   *k*PG*w.d*

The value of *k* must not be greater than *d* + 1, where *d* is the number of digits in the **E***w.d*, **D***w.d*, or **G***w.d* output fields.

Scale Factor

The scale factor, *k*, determines the appropriate editing as follows:

- For input with **F**, **E**, **D**, and **G** editing (provided there is no exponent in the field) and **F** output editing, the magnitude represented by the external field equals the magnitude of the internal value multiplied by 10k.
- For input with **F**, **E**, **D**, and **G** editing containing a real exponent, the scale factor has no effect.
- For output with **E** and **D** editing, the basic value part is multiplied by 10k and the real exponent is reduced by *k.*

- For output with **G** editing, the scale factor has no effect unless the data to be edited is outside the range that permits **F** editing. If the use of **E** editing is required, the effect of the scale factor is the same as **E** output editing. (See Real Type in Chapter 2.)

On input, if no exponent is given, the scale factor in any of the above field descriptors multiplies the data by 10^{-k} and assigns it to the corresponding I/O list element. For example, a 2**P** scale factor multiplies an input value by .01. A −2**P** scale factor multiplies an input value by 100. However, if the external field contains an explicit exponent, the scale factor has no effect. Table 9−18 gives examples of scale factors.

Table 9−18 Scale Factor Examples

Format	External Field	Internal Value
3pe10.5	" 37.614"	.37614E−01
3pe10.5	" 37.614E2"	.37614E+04
−3pe10.5	" 37.614"	.37614e+05

On output, the effect of the scale factor depends on the type of field descriptor associated with it.

For the **F** field descriptor, the value of the I/O list element is multiplied by 10k before transfer to the external record: a positive scale factor moves the decimal point to the right; a negative scale factor moves the decimal point to the left. The value represented is 10k multiplied by the internal value.

For output with the **E** or **D** field descriptor, the basic real constant part of the external field is multiplied by 10k and the exponent is reduced by *k*. The value represented is unchanged. A positive scale factor moves the decimal point to the right and decreases the exponent; a negative scale factor moves the decimal point to the left and increases the exponent. In summation,

$k > 0$	moves the decimal point *k* digits to the right.
$k < 0$	moves the decimal point *k* digits to the left.
$k = 0$	leaves the decimal point unchanged.

Table 9−19 provides scale format output examples.

Table 9−19 Scale Format Output Examples

Format	Internal Value	External Field
1pe12.3	−.270139E+03	" 2.701E+0 2"
1pe12.2	−270139E+03	" 2.70E+02"
−1pe12.2	−.270139E+03	" 0.03E+04"

On output, the effect of the scale factor for the **G** field descriptor is suspended if the magnitude of the output data is within the range permitting **F** editing because the **G** field descriptor supplies its own scaling function. The **G** field descriptor functions as an **E** field descriptor if the magnitude of the data value is outside its range. In this case, the scale factor has the same effect as the **E** field descriptor.

On output under **F** field descriptor control, a scale factor actually alters the magnitude of the value represented, multiplying or dividing it by ten. On output, a scale factor under **E**, **D**, or **G** field descriptor control merely alters the form in which the value is represented.

If you do not specify a scale factor with a field descriptor, a scale factor of zero is assumed at the beginning of the execution of the statement. Once a scale factor is specified, it applies to all subsequent **F**, **E**, **D**, and **G** field descriptors in the same format specification, unless another scale factor appears. A scale factor of zero can be reinstated only with an explicit **P** specification.

L Edit Descriptor

The **L** edit descriptor is used for logical data. The specified I/O list item must be of type **LOGICAL**. It has the form

`L`*w*

where *w* is a nonzero, unsigned integer constant denoting field width.

For input, the field must consist of optional blanks followed by an optional decimal point followed by a **T** (for true) or **F** (for false). The **T** or **F** can be followed by additional characters that have no effect. The logical constants **.TRUE.** and **.FALSE.** are acceptable input forms.

For output, the field consists of *w* - 1 blanks followed by a **T** or an **F**, for true and false, respectively, according to the value of the internal data. Table 9−20 shows **L** field examples.

Table 9−20 L Field Examples

Format	Internal Value	External Field
L5	.TRUE.	" T"
l1	.FALSE.	"F"

The **L** edit descriptor can also be used to process integer data items. All nonzero values are displayed as **.TRUE.** and all zero values as **.FALSE.**.

A Edit Descriptor

The **A** edit descriptor is used for editing character or Hollerith data. It has the form

`A[`*w*`]`

where *w* is a nonzero, unsigned integer constant denoting the width, in number of characters, of the external data field. If *w* is omitted, the size of the I/O list item determines the length *w*.

The corresponding I/O list item can be any data type. If it is character data type, character data is transmitted. If it is any other data type, Hollerith data is transmitted.

In an input statement, the **A** edit descriptor reads a field of *w* characters from the record without interpretation and assigns it to the corresponding I/O list item. The maximum number of characters that can be stored depends on the size of the I/O list item. For character I/O list elements, the size is the length of the character variable, character substring reference, or character array element. For numeric and logical I/O list elements, the size depends on the data type, as shown in Table 9−21

Table 9−21 I/O List Element Sizes

I/O List Element	Maximum Number of Characters
LOGICAL*1	1
LOGICAL*2	2

LOGICAL*4	4
INTEGER*2	2
INTEGER*4	4
REAL*4 (REAL)	4
REAL*8 (DOUBLE PRECISION)	8
COMPLEX*8 (COMPLEX)	8
COMPLEX*16 (DOUBLE COMPLEX)	16

If *w* is greater than the maximum number of characters that can be stored in the corresponding I/O list item, only the right most characters of the field are assigned to that element. The left most excess characters are ignored. If *w* is less than the number of characters that can be stored, *w* characters are assigned to the list item and left justified, and trailing blanks are added to fill it to its maximum size.

Input Example

Table 9−22lists **A** field input examples.

Table 9−22 A Field Input Examples

Format	External Field	Internal Value	Representation
A6	"FACE #"	"#"	(CHARACTER*1)
A6	"FACE #"	"E #"	(CHARACTER*3)
A6	"FACE #"	"FACE #"	(CHARACTER*6)
A6	"FACE #"	"FACE # "	(CHARACTER*8)
A6	"FACE #"	"#"	(LOGICAL*1)
A6	"FACE #"	"#"	(INTEGER*2)
A6	"FACE #"	"CE #"	(REAL*4)
A6	"FACE #"	"FACE # "	(REAL*8)

In an output statement, the **A** field descriptor writes the contents of the corresponding I/O list item to the record as an external field *w* characters long. If *w* is greater than the list item size, the data appears in the field, right justified, with leading blanks. If *w* is less than the list element, only the left most *w* characters from the I/O list item are transferred.

Table 9−23lists **A** field output examples.

Table 9−23A Field Output Examples

Format	Internal Value	External Field
A6	"GREEK"	" GREEK"
A6	"FRENCH"	"FRENCH"
A6	"PORTUGUESE"	"PORTUG"

If you omit *w* in an **A** field descriptor, a default value is supplied based on the data type of the I/O list item. If it is character type, the default value is the length of the I/O list element. If it is numeric or logical data type, the default value is the maximum number of characters that can be stored in a variable of that data type as described for input.

Repeat Counts

The **I**, **O**, **Z**, **F**, **E**, **D**, **G**, **L**, and **A** field descriptors can be applied to a number of successive I/O list items by preceding the field descriptor with an unsigned integer constant, called the repeat count. For example, 4**F**5.2 is equivalent to **F**5.2, **F**5.2, **F**5.2, **F**5.2.

Enclosing a group of field descriptors in parentheses, and preceding the enclosed group with a repeat count, repeats the entire group. Thus, 2(I6,**F**8.4) is equivalent to **I**6,**F**8.4,**I**6,F8.4.

H Field Descriptor

The **H** field descriptor is used for output of character literal data. It has the form:

*n*H*xxx*… *x*

where

n	is an unsigned integer constant denoting the number of characters that comprise the character literal.
x	comprises the character literal and consists of *n* characters, including blanks.

In an output statement, the **H** field descriptor writes the *n* characters following the letter **H** from the field descriptor to the record as an external field *n* characters long. The **H** field descriptor does not correspond to an output list item.

Table 9−24lists examples of **H** edit description output.

Table 9−24H Edit Description Output Examples

Specification	External Field
6HAb CdE	Ab CdE
1H9	9
4H'a2'	'a2'

An **H** field descriptor must not be encountered by a **READ** statement.

Character Edit Descriptor

A character edit descriptor has one of the following forms:

'X1 X2 … Xn'
X1 X2 … Xn

where *X1 X2 … Xn* are members of the Fortran character set forming a valid character literal. The width of the output field is the number of characters contained in the character literal, excluding the enclosing apostrophes or quotation marks. The character edit descriptor does not correspond to an output list item. Within a character edit descriptor delimited by apostrophes, an apostrophe is represented by two successive apostrophe characters. Within a character edit descriptor delimited by quotation marks, a quotation mark is represented by two successive quotation mark characters.

Example

Table 9−25lists character edit description examples.

Table 9–25 Character Edit Description Examples

Output Specification	External Field
’sum =’	sum =
.sum =	sum =
.don’t	don’t
’here’’s the answer’	here’s the answer
’he said, "yes"’	he said, "yes"
.he said, ""yes""	he said, "yes"

A character edit descriptor must not be encountered by a **READ** statement.

Use of quotation marks as a character edit descriptor is an enhancement to Fortran 77.

Q Edit Descriptor

The **Q** edit descriptor is used to determine the number of characters remaining to be read from the current input record. It has the form

```
Q
```

When a **Q** descriptor is encountered during the execution of an input statement, the corresponding input list item must be type integer. Interpretation of the **Q** edit descriptor causes the input list item to be defined with a value that represents the number of character positions in the formatted record remaining to be read. Therefore, if *c* is the character position within the current record of the next character to be read and the record consists of *len* characters, then the item is defined with the value

n = max (*len* – *c* + 1, 0)

If no characters have yet been read, then *n* = *len*, the length of the record. If all the characters of the record have been read (*c* > *len*), then *n* is zero.

The **Q** edit descriptor must not be encountered during the execution of an output statement.

Input Example

The following is an example of **Q** edit description input:

```
     INTEGER N
     CHARACTER LINE * 80
     READ (5, 100) N, LINE (1:N)
100  FORMAT (Q, A)
```

Edit Descriptor Reference

After each **I**, **O**, **Z**, **F**, **E**, **D**, **G**, **L**, **A**, **H**, or character edit descriptor is processed, the file is positioned after the last character read or written in the current record.

The **X**, **T**, **TL**, and **TR** descriptors specify the position at which the next character will be transmitted to or from the record. They do not change any characters in the record already written or by themselves affect the length of the record.

If characters are transmitted to positions at or after the position specified by a **T**, **TL**, **TR**, or **X** edit descriptor, positions skipped and not previously filled are filled with blanks.

X Edit Descriptor

The **X** edit descriptor specifies a position forward (to the right) of the current position. It is used to skip characters on the external medium for input and output. It has the form

n`X`

where *n* is a nonzero, unsigned integer constant denoting the number of characters to be skipped.

T Edit Descriptor

The **T** edit descriptor specifies an absolute position in an input or output record. It has the form:

`T`*n*

where *n* indicates that the next character transferred to or from the record is the *n*th character of the record.

TL Edit Descriptor

The **TL** edit descriptor specifies a position to the left of the current position. It has the form

`TL`*n*

where *n* indicates that the next character to be transferred from or to the record is the *n*th character to the left of the current character. The value of *n* must be greater than or equal to one.

If *n* is the current character position, then the first character in the record is specified.

TR Edit Descriptor

The **TR** edit descriptor specifies a position to the right of the current position. It has the form

`TR`*n*

where *n* indicates that the next character to be transferred from or to a record is the *n*th character to the right of the current character. The value of *n* must be greater than or equal to one.

BN Edit Descriptor

The **BN** edit descriptor causes the processor to ignore blank characters in a numeric input field and to right justify the remaining characters, as though the blanks that were ignored were leading blanks. It has the form

`BN`

The **BN** descriptor affects only **I**, **O**, **Z**, **F**, **E**, **D**, and **G** editing and then only on input fields.

BZ Edit Descriptor

The **BZ** edit descriptor causes the processor to treat all the embedded and trailing blank characters it encounters within a numeric input field as zeros. It has the form:

```
BZ
```

The **BZ** descriptor affects only **I**, **O**, **Z**, **F**, **E**, **D**, and **G** editing and then only on input fields.

SP Edit Descriptor

The **SP** edit descriptor specifies that a plus sign be inserted in any character position that normally contains an optional plus sign and whose actual value is \xb3 0. It has the form

```
SP
```

The **SP** descriptor affects only **I**, **F**, **E**, **D**, and **G** editing and then only on output fields.

SS Edit Descriptor

The **SS** edit descriptor specifies that a plus sign should *not* be inserted in any character position that normally contains an optional plus sign. It has the form

```
SS
```

The **SS** descriptor affects only **I**, **F**, **E**, **D**, and **G** editing and then only on output fields.

S Edit Descriptor

The **S** edit descriptor resets the option of inserting plus characters (+) in numeric output fields to the processor default. It has the form

```
S
```

The **S** descriptor counters the action of either the **SP** or the **SS** descriptor by restoring to the processor the discretion of producing plus characters (+) on an optional basis. The default is to **SS** processing; the optional plus sign is not inserted when **S** is in effect.

The **S** descriptor affects only **I**, **F**, **E**, **D**, and **G** editing and then only on output fields.

Colon Descriptor

The colon character (:) in a format specification terminates format control if no more items are in the I/O list. The colon descriptor has no effect if I/O list items remain.

$ Edit Descriptor

The $ edit descriptor suppresses the terminal line−mark character at the end of the current output record. It has the form

```
$
```

The $ descriptor is nonrepeatable and is ignored when encountered during input operations.

Output Example

```
      print 100, 'enter a number:'
100   format (1x, a, $)
      read *, x
```

Complex Data Editing

A complex value consists of an ordered pair of real values. If an **F**, **E**, **D**, or **G** field descriptor is encountered, and the next I/O list item is complex, then the descriptor is used to edit the real part of the complex item. The next field descriptor is used to edit the imaginary part.

If an **A** field descriptor is encountered on input or output, and the next I/O list item is complex, then the **A** field descriptor is used to translate Hollerith data to or from the external field and the entire complex list item. The real and imaginary parts together are treated as a single I/O list item.

In an input statement with **F**, **E**, **D**, or **G** field descriptors in effect, the two successive fields are read and assigned to a complex I/O list element as its real and imaginary parts, respectively.

Table 9–26contains examples of complex data editing input.

Table 9–26 Complex Data Editing Input Examples

Format	External Field	Internal Value
f8.5,f8.5	"1234567812345.67"	(.12345678E+03,.1234567E+05)
f9.1,f9.3	"734.432E8123456789"	(.734432E+11,.123456789E+06)

In an output statement with **F**, **E**, **D**, or **G** field descriptors in effect, the two parts of a complex value are transferred under the control of successive field descriptors. The two parts are transferred consecutively, without punctuation or spacing, unless the format specification states otherwise.

Table 9–27contains examples of complex data editing output.

Table 9–27 Complex Data Editing Output Examples

Format	Internal Value	External Field
2f8.5	(.23547188E+01,.3456732E+01)	"2.35472 3.45673"
e9.2,",",e5.3	(.47587222E+05,.56123E+02)	"0.48E+06, *****"

Carriage Control

A formatted record can contain a prescribed carriage–control character as the first character of the record. The carriage–control character determines vertical spacing in printing when the**CARRIAGECONTROL** keyword of the **OPEN** statement is set to **FORTRAN** (as described in ‘‘OPEN’’ of Chapter 8, "Input/Output Statements.") Table 9–28lists the carriage–control characters.

Table 9–28 Carriage–Control Characters

Character	Effect on Spacing
Blank	Single space
0	Double space
1	To first line of next page

+	No vertical spacing
$	Output starts at the beginning of the next line; carriage return at the end of the line is suppressed
ASCII NUL	Overprints with no advance; does not return to the left margin after printing

The carriage−control character is not printed, and the remaining characters, if any, are printed on one line beginning at the left margin. If there are no characters in the record, the vertical spacing is one line and no characters will be printed in that line.

Slash Editing

A slash (/) placed in a format specification terminates input or output for the current record and initiates a new record. For example

```
    WRITE (6,40) K,L,M,N,O,P
40  FORMAT (3I6.6/I6,2F8.4)
```

is equivalent to

```
    WRITE (6,40) K,L,M
40  FORMAT (3I6.6)
    WRITE (6,50) N,O,P
50  FORMAT (I6,2F8.4)
```

On input from a sequential−access file, the current portion of the remaining record is skipped, a new record is read, and the current position is set to the first character of the record. *n* slashes in succession cause *n* - 1 records to be skipped.

On output to a file connected for sequential access, a new record is created and becomes the last and current record of the file. Also, *n* slashes in succession cause *n* - 1 blank lines to be generated.

Through the use of two or more successive slashes in a format specification, entire records can be skipped for input and records containing no characters can be generated for output. If the file is an internal file, or a file connected for direct access, skipped records are filled with blank characters on output.

n slashes at the beginning or end of a format specification result in *n* skipped or blank records. On input and output from a direct−access file, the record number is increased by one and the file is positioned at the beginning of the record that has that record number. This record becomes the current record.

Interaction Between I/O List and Format

The beginning of formatted data transfer using a format specification initiates format control. Each action of format control depends on information jointly provided by

- the next descriptor contained in the format specification
- the next item in the I/O list, if one exists

If an I/O list specifies at least one list item, at least one repeatable descriptor must exist in the format specification. Note that an empty format specification of the form ( ) can be used only if no list items are specified; in this case, one input record is skipped or one output record containing no characters is written.

Except for a field descriptor preceded by a repeat specification, *r ed*, or a format specification preceded by a repeat specification, *r (flist)*, a format specification is interpreted from left to right (see ‘‘Repeat Counts’’). Note that an omitted repeat specification is treated the same as a repeat specification whose value is one.

To each repeatable field descriptor interpreted in a format specification, there corresponds one item specified by the I/O list, except that a list item of type complex is treated as two real items when an **F**, **E**, **D**, or **G** field descriptor is encountered. To each **P**, **X**, **T**, **TL**, **TR**, **S**, **SP**, **SS**, **H**, **BN**, **BZ**, slash (/), colon (:), dollar sign ($), or character edit descriptor, there is no corresponding item specified by the I/O list, and format control communicates information directly to the record.

Whenever format control encounters a repeatable edit descriptor in a format specification, it determines whether there is another item in the I/O list. If there is such an item, it transmits appropriately edited information between the item and the record, and then format control proceeds. If there is no other item, format control terminates.

If format control encounters the right most parenthesis of a complete format specification and no items remain in the list, format control terminates. However, if there are more items in the list, the file is positioned at the beginning of the next record, and format control then reverts to the beginning of the format specification terminated by the last preceding right parenthesis ()). If there is no such preceding right parenthesis ()), format control reverts to the first left parenthesis (() of the format specification. If such a reversion occurs, the reused portion of the format specification must contain at least one repeatable edit descriptor. If format control reverts to a parenthesis that is preceded by a repeat specification, the repeat specification is reused. Reversion of format control, of itself, has no effect on the scale factor (see ‘‘D Field Descriptor’’) the **S**, **SP**, or **SS** edit descriptor sign control, or the **BN** or **BZ** edit descriptor blank control.

List−Directed Formatting

List−directed formatting allows formatted input and output without specifying a format specification. An asterisk (*) is used as a format identifier to invoke a list−directed format.

List−directed formatting can be applied to both internal and external files.

List−Directed Input

The characters in one or more list−directed records form a sequence of values and value separators. Each value is either a constant, or a null value or has one of the following forms:

*r*c*
*r**

where

r	is a nonzero, unsigned integer constant denoting the number of successive appearances of *c* or null values.

c is a constant.

The *r** form is equivalent to *r* successive null values. Neither form can contain embedded blanks, except where permitted within the constant *c*.

Data values can be separated with one of the following value separators:

- A comma optionally preceded and followed by one or more contiguous blanks.
- A slash (/) optionally preceded and followed by one or more contiguous blanks. A slash encountered by a list−directed input statement ends the execution of the input statement after assignment of the previous value, if any; any remaining list items are treated as if null values were supplied. A slash is not used as a separator on output.
- One or more contiguous blanks between two constants or following the last constant. Blanks used in the following manner are not treated as part of any value separator in a list−directed input record:
  - blanks within a character constant
  - embedded blanks surrounding the real or imaginary part of a complex constant
  - leading blanks in the first record read by each execution of a list−directed input statement, unless immediately followed by a slash or comma

The end of a record has the effect of a blank, except when it appears within a character constant. Two or more consecutive blanks are treated as a single blank, unless they occur within a character constant.

There are three differences between the input forms acceptable to format specifiers for a data type and those used for list−directed formatting. A data value must have the same type as the list item to which it corresponds. Blanks are not interpreted as zeros. Embedded blanks are only allowed in constants of character or complex type.

Rules governing input forms of list items for list−directed formatting are

- For data of type real or double precision, the input form is the same as a numeric input field for **F** editing that has no fractional part, unless a decimal point appears within the field.
- For data of type complex, the input form consists of an ordered pair of numeric constants separated by a comma and enclosed in a pair of parentheses. The first numeric constant is the real part of the complex value, while the second constant is the imaginary part. Each of the constants representing the real and imaginary parts may be preceded or followed by blanks. The end of a record may occur between the real part and the comma or between the comma and the imaginary part.
- For data of type logical, the input form must not include either slashes or commas among the optional characters allowed for **L** editing.
- For data of type character, the input form is a character constant: a non empty string of characters enclosed in apostrophes or quotation marks. When apostrophes are used as the character constant delimiter, each apostrophe within the apostrophes is represented by a pair of apostrophes without an intervening blank or end of record.

  When quotation marks are used as the character constant delimiter, each quotation mark within the quotation marks is represented by a pair of quotation marks without an intervening blank or end of record. Character constants can be continued on as many records as needed. Constants are assigned to list items as in character assignment statements.

- A null value is specified by two successive value separators, by the *r** form, or by not having any characters before the first value separator in the first record read by the execution of the list–directed statement. A null value has no effect on the corresponding list item. A single null value may represent an entire complex constant but cannot be used as either the real or imaginary part alone.
- You can specify commas as value separators in the input record when executing a formatted read of non character variables. The commas override the field lengths in the input statement. For example,

  ```
  (i10, f20.10,i4)
  ```

  reads the following record correctly:

  ```
  −345,.05e−3,12
  ```

List–Directed Output

The form of the values produced is the same as that required for input, except as noted below:

- Logical output constants are **T** for the value true and **F** for the value false.
- Integer output constants are produced as for an **I***w* edit descriptor, where *w* depends on whether the list item is **INTEGER*2** or **INTEGER*4** type.
- For complex constants, the end of a record will occur between the comma and the imaginary part only if the entire constant is as long as, or longer than, an entire record.
- Produced character constants are not delimited by apostrophes or quotation marks, are not preceded or followed by a value separator, and have each internal apostrophe represented externally by one apostrophe and each internal quotation mark represented by one quotation mark. A blank character for carriage control is inserted at the beginning of a record containing the continuation of a character constant.
- Slashes and null values are not produced, but each record begins with a blank character to provide carriage control if the record is printed.
- Two noncharacter values in succession in the same record will be separated by a value separator of one or more blanks. No value separator is produced before or after a character value.