Chapter 15

Input/Output Editing

Input/output editing directs the conversion between an internal representation of data and data fields embedded in formatted records of a file. As indicated in Section 13.4.1.2, formatted I/O statements read and write records that are formatted either according to a specification provided by the program, or according to the rules of list-directed formatting. A `FMT=` specifier that is a label, a scalar integer variable, or a character expression specifies editing under control of a *format specification*, and an asterisk (`*`) specifies list-directed formatting. A label or a scalar integer variable must reference a `FORMAT` statement specifying a format specification, and a character expression must form a valid format specification.

The form of a format specification is given in the next section, and the three classes of edit descriptors that may be used in a format specification are explained in Sections 15.2, 15.3, and 15.4. Section 15.5 explains more about the interaction between an I/O list and a format specification. List-directed formatting is described in Section 15.6.

15.1 Format Specification

A *format specification* (R15-1) provides editing instructions to be performed during data transfer.

A *format-specification* is R15-1
(*edit-descriptor-list*)

An *edit-descriptor-list* is: R15-2
[edit-descriptor [, edit-descriptor]...]

- The comma used to separate elements of an *edit-descriptor-list* (R15-2) may be omitted in some cases (see notes in Sections 15.3.2, 15.3.3, and 15.3.6).

Copyright © 1989–1994 Thinking Machines Corporation

An *edit-descriptor* is one of: R15-3

- *[r] data-edit-desc*
- *control-edit-desc*
- *char-literal-edit-desc*
- *[r]* (*edit-descriptor-list*)

An *r* is: R15-4

integer-literal-constant

♦ *r* must be positive.

An *edit descriptor* (R15-3) of a format specification is one of three classes — data, control, or character string — or is a group of edit descriptors enclosed in parentheses. An integer literal constant *r* prefixing a data edit descriptor or an edit descriptor group is called a *repeat specification*.

The data edit descriptors direct the editing of data fields in a record, causing conversion of an I/O list data item to or from its internal representation. Control edit descriptors affect the transfer of data or the conversions performed by subsequent data edit descriptors. Character literal edit descriptors are literal strings that provide text to be output. Data edit descriptors are described in Section 15.2, control edit descriptors in Section 15.3, and character literal edit descriptors in Section 15.4. Table 24, page 285, summarizes the forms of edit descriptors available in a format specification.

The following are examples of format specifications:

```
( I5, 4( 2P, G15.8, 1X, I2 ) )
( I5, /, I5 )
( L1, :, 1X, G15.2, 2I6 )
( ' Absolute:', T12, SP, I5 )
```

15.1.1 FORMAT Statement

A `FORMAT` statement (R15-5) associates a label with a format specification (R15-1).

A *format-stmt* is: R15-5

label `FORMAT` *format-specification*

As indicated in Section 13.4.1.2, two forms of a `FMT` I/O control specifier (R13-8) use labeled `FORMAT` statements:

- a *label*, specifying that formatted I/O is to be performed under control of the `FORMAT` statement specified by the label

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

- a scalar integer variable, indicating that formatted I/O is to be performed under control of a FORMAT statement whose label has been assigned to the variable using the ASSIGN statement.

Examples of FORMAT statements:

```
10    FORMAT( 1P E12.4, I10 )
20    FORMAT( I12 / 'Dates: ', 2( 2I3, I5 ) )
```

The first statement contains a control edit descriptor and two data edit descriptors. The second statement contains a data edit descriptor, a control edit descriptor, a character literal edit descriptor, and a group of two data edit descriptors prefixed by a repeat specification.

15.1.2 Character String Format Specification

A character expression as a FMT= I/O control specifier (R13-8) in an I/O statement causes the value of the character expression itself to be used as a format specification, as indicated in Section 13.4.1.2.

All character positions up to and including the final right parenthesis of the format specification must be defined when the input/output statement is executed, and must not be modified during the execution of the statement. (Character positions following the right parenthesis that ends the format specification need not be defined.)

If a FMT= specifier identifies a character array entity, it is treated as if all the elements of the array were concatenated in array element order. However, if a FMT= specifier refers to a character array element, the entire specification must be contained within that array element.

The following four formatted WRITE statements are identical in effect. The first three use character string format specifications, and the last uses the format specification in a FORMAT statement.

```
      CHARACTER*(*), PARAMETER :: FMTC = '1PE12.4, I10'
      CHARACTER*8 :: FMTA(6)
      DATA FMTA / '(', ',', '1PE12.4', ',', 'I10', ')' /

      WRITE( *, '( 1PE12.4, I10 )' ) X, I ! char literal
      WRITE( *, '(' // FMTC // ')' ) X, I ! char expression
      WRITE( *, FMTA ) X, I         ! character array name
      WRITE( *, 10 ) X, I           ! format statement
10    FORMAT( 1PE12.4, I10 )
```


Copyright © 1989–1994 Thinking Machines Corporation

In the example, the named character constant FMTC is used to form a valid format specification in the second WRITE statement; the whole character array FMTA forms a valid format specification in the third WRITE statement when its elements are concatenated in array element order.

15.2 Data Edit Descriptors

A *data edit descriptor* (R15-6) directs the editing of a data field in a record. A *data field* is a substring of the characters in a record; a data field has a *field width,* which is its size, in characters. A data edit descriptor may be preceded by a repeat specification *r* in a format specification.

A *data-edit-desc* is one of: R15-6

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

Each of *w*, *m*, *d*, and *e* is an: R15-6
integer-literal-constant

- *w* and *e* must be positive and *d* and *m* must be zero or positive.
- The value of *m*, *d*, and *e* may be restricted further by the value of *w*. The codes I, B, O, Z, F, E, G, L, A, and D indicate the manner of editing.

On input, a variable in the I/O list of a file input statement becomes defined with the value represented by its associated data field unless an error or an end-of-file condition occurs. On output, the I/O list expression is evaluated before transfer to the data field of a record.

In the following sections, ± represents a plus or a minus character, *blanks* represents a string of one or more blanks, *digits* represents a string of one or more decimal digits (unless otherwise specified), and a subscripted *x*, *y*, or *z* represents a single decimal digit.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

15.2.1 Numeric Editing

The I, B, O, Z, F, E, D, and G edit descriptors may be used to edit numeric data: the I, B, O, and Z edit descriptors are used for integer data, and the others are used for real and complex data, of either single or double precision.

For numeric output, if the number of characters produced exceeds the field width or if an exponent exceeds its specified length using the E*w*.*d*E*e* or G*w*.*d*E*e* edit descriptor, the entire field of width *w* is filled with asterisks. However, the field is not filled with asterisks if omitting the optional characters causes the field width not to be exceeded. (Note that when an SP edit descriptor is in effect, a leading plus is not an optional character.)

15.2.1.1 Integer Editing (I Editing)

The I*w* and I*w*.*m* edit descriptors indicate that an integer field to be edited occupies *w* positions. The corresponding I/O list item must be of type integer. Examples of the effects of integer editing are given in Table 12, page 262.

Input: I*w* (and I*w*.*m*) edit descriptors

An integer field to be edited must have the form:

[blanks] [±] digits

A leading minus causes the negative of the value represented by the decimal *digits* to be assigned to the I/O list variable, and a leading plus has no effect. The value of *m* has no effect on input.

Leading blanks are ignored. Any embedded or trailing blanks are either taken as zeros or ignored, depending on blank control in effect for the statement (see Section 15.3.7). A field containing only blanks is considered to be zero.

Output: I*w* edit descriptor

An integer value is output in the following form when edited using the I*w* edit descriptor:

[blanks] [±] digits

The output field for the I*w* edit descriptor is right-justified and may contain leading blanks. It contains a minus if the value of the data item is negative, or an optional plus if the value is positive (the output of a leading plus may be forced by the SP edit descriptor). The magnitude of the value is represented as an unsigned string of one or more decimal digits without leading zeros.

Copyright © 1989–1994 Thinking Machines Corporation

Output: I*w*.*m* edit descriptor

An integer value is output in the following form when edited using the I*w*.*m* edit descriptor:

[blanks] [±] [zeros] digits

The output field for the I*w*.*m* edit descriptor is the same as for the I*w* edit descriptor, except that the string of decimal digits consists of at least *m* digits: sufficient leading zeros are included to achieve the minimum of *m* digits, if necessary.

The value of *m* must not exceed the value of *w*. If *m* is zero and the value of the data item is zero, the output field consists of only blank characters, regardless of the sign control in effect for the statement (see Section 15.3.5).

Table 12. Effects of I/O editing using I edit descriptors.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 12345678901234567890
I5	123	123	123
	-123	-123	-123
I5.5	123	123	00123
	21	21	00021
	-21	-21	*****
I10.8	12345	12345	00012345
	-32768	-32768	-00032768

15.2.1.2 Integer Editing: Binary (B Editing), Octal (O Editing) or Hexadecimal (Z Editing)

The B*w*, B*w*.*m*, O*w*, O*w*.*m*, Z*w*, and Z*w*.*m* edit descriptors indicate that an unsigned integer field to be edited occupies *w* positions, in binary (base 2) representation for the B forms, in octal (base 8) representation for the O forms, or in hexadecimal (base 16) representation for the Z forms. The corresponding I/O list item must be of type integer. Examples of the effects of binary, octal, and hexadecimal integer editing are given in Table 13, p. 264.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Input: `B`*w* (and `B`*w*`.`*m*) edit descriptors, `O`*w* (and `O`*w*`.`*m*) edit descriptors or `Z`*w* (and `Z`*w*`.`*m*) edit descriptors

A binary, octal, or hexadecimal integer field to be edited must have the form:

[blanks] digits

where *digits* represents a string of one or more binary digits (the characters `0` and `1`) octal digits (the characters `0` through `7`), or hexadecimal digits (the decimal digits `0` through `9` and the characters `A` through `F`, either lower- or uppercase). The value of *m* has no effect on input.

Leading blanks are ignored. Any embedded or trailing blanks are either taken as zeros or ignored, depending on blank control in effect for the statement (see Section 15.3.7). A field containing only blanks is considered to be zero.

Output: `B`*w*, `O`*w*, or `Z`*w* edit descriptors

An integer value is output in the following form when edited using the `B`*w*, `O`*w*, or `Z`*w* edit descriptors:

[blanks] digits

The output field for the `B`*w*, `O`*w*, and `Z`*w* edit descriptors is right-justified and may contain leading blanks. The magnitude of the value is represented by *digits* as an unsigned string of one or more binary, octal, or hexadecimal digits without leading zeros.

Output: `B`*w*`.`*m*, `O`*w*`.`*m*, or `Z`*w*`.`*m* edit descriptors

An integer value is output in the following form when edited using the `B`*w*`.`*m*, `O`*w*`.`*m*, or `Z`*w*`.`*m* edit descriptors:

[blanks] [zeros] digits

The output field for the `B`*w*`.`*m*, `O`*w*`.`*m*, or `Z`*w*`.`*m* edit descriptors are the same as for the `B`*w*, `O`*w*, or `Z`*w* edit descriptors, except that the string of binary, octal, or hexadecimal digits consists of at least *m* digits: sufficient leading zeros are included to achieve the minimum of *m* digits, if necessary.

The value of *m* must not exceed the value of *w*. If *m* is zero and the value of the data item is zero, the output field consists of only blank characters.

Copyright © 1989–1994 Thinking Machines Corporation

Table 13. . Effects of I/O editing using B, O, and Z edit descriptors

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 123456789012345678901
B5	1001	9	1001
	101	5	101
B10.5	1001	9	01001
	101	5	00101
O6	123	83	123
	7777	4095	7777
	10000	4096	10000
O11.5	37777777777	-1	37777777777
	21	17	00021
	37777777770	-8	37777777770
Z4	ff	255	FF
	a	10	A
Z8.4	f	15	000F
	10000	65536	10000
	ffffffff	-1	FFFFFFFF
Z10.8	f f	3855	00000F0F
	1000	4096	00001000
	00000FF	255	000000FF

15.2.1.3 Real and Complex Editing (F, E, and D Editing)

The F, E, D, and G edit descriptors specify the editing of real and complex data. An I/O list item associated with one of these edit descriptors must be of type real or complex. The effect of the G edit descriptor is defined in terms of the F and the E edit descriptors in Section 15.2.1.4.

15.2.1.3.1 Decimal Notation (F Editing) The F*w*.*d* edit descriptor indicates that a field to be edited occupies *w* positions, the fractional part of which contains *d* digits. The value of *d* must not exceed the value of *w*. Examples of the effects of F editing are given in Table 14, page 267.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Input: `F`*w*`.`*d* edit descriptor

A real field to be edited must have one of the following four forms:

[blanks] [±] digits [. digits]
[blanks] [±] digits [. digits] `E` *[blanks] [±] digits*
[blanks] [±] digits [. digits] `D` *[blanks] [±] digits*
[blanks] [±] digits [. digits] ± *[blanks] digits*

The first form consists of a *significand part* only. The last three forms consist of a significand part followed by an *exponent part* beginning with an `E`, a `D`, or a nonoptional sign. The significand is composed of a *magnitude part* (the first string of *digits*), and an optional *fractional part* (the *digits* following the decimal point).

A leading minus causes the negative of the value represented by the significand part to be assigned to the input list variable, and a leading plus has no effect. The *d* has no effect if the field contains a decimal point. If the decimal point is omitted, the rightmost *d* digits $x_1x_2...x_d$ of the string, with leading zeros assumed if necessary, are interpreted as the fractional part of the value represented. The string of digits may contain more digits than are used internally to approximate the value of the constant.

Leading *blanks* are ignored prior to the optional sign (±) in the significand part or before the first digit in the exponent part. Any blank characters after the optional sign (±) or interspersed with the *digits* in either the significand part or the exponent part are either taken as zeros or ignored, depending on blank control in effect for the statement (see Section 15.3.7). A field containing only blanks is considered to be zero.

If the input field does not contain an exponent (as in the first form above), the effect is as if that form were followed by an exponent with a value of –*k*, where *k* is the established scale factor (15.3.6). (The default value of `0` for *k* has no effect.)

All three forms of exponent parts are processed identically. The *digits* of the exponent are taken as a power of ten by which the significand part is to be multiplied to obtain the value represented.

Output: `F`*w*`.`*d* edit descriptor

A real value is output in the following form when edited using the `F`*w*`.`*d* edit descriptor:

[blanks] [±]$x_1x_2...x_{w-d-1}$. $y_1y_2...y_d$

where the x_i and y_i represent decimal digits.

Copyright © 1989–1994 Thinking Machines Corporation

The representation is right-justified in the field and may contain leading *blanks*. The field contains a leading minus if the output value is negative, or an optional leading plus if the value is positive (a leading plus may be forced using the `SP` edit descriptor). The magnitude of the value (represented by the digits x_i) and the fractional part of the value (represented by the y_i) are separated by a decimal point, and either string may be empty.

The string of digits represents the magnitude of the internal value as modified by the established scale factor (see Section 15.3.6) and rounded to *d* fractional digits. Leading zeros are not output except for an optional zero immediately to the left of the decimal point if the magnitude of the value in the output field is less than one. A zero appears as the magnitude if there would otherwise be no digits in the output field.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Table 14. Effects of I/O editing using `F` edit descriptors.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 12345678901234567890
F6.0	1234	1234.00	1234.
	12345	12345.0	12345.
	.5	0.500000	1.
	.4	0.400000	0.
	-33	-33.0000	-33.
	+55	55.0000	55.
F5.2	1	1.000000E-02	0.01
	12	0.120000	0.12
	123	1.23000	1.23
	1234	12.3400	12.34
	12345	123.450	*****
	1.	1.00000	1.00
	12.	12.0000	1.20
	123.	123.000	*****
F12.6	123456789	123.457	123.456789
	.123456789	0.123457	0.123457
	1e4	1.000000E-02	0.010000
	1+4	1.000000E-02	0.010000
	1.e4	10000.0	10000.000000
	1.+4	10000.0	10000.000000
F20.12	123456789012345	123.457	123.456789012345
	1.e5	100000.	100000.000000000000
	1.e-11	1.000000E-11	0.000000000010
	1e5	1.000000E-07	0.000000100000
	1e-11	1.000000E-23	0.000000000000
	0	0.000000E+00	0.000000000000

15.2.1.3.2 Exponential Notation (E and D Editing). The `E`*w*`.`*d*, `D`*w*`.`*d*, and `E`*w*`.`*d*`E`*e* edit descriptors indicate that a field to be edited occupies *w* positions containing a fractional part consisting of *d* digits (unless a scale factor greater than one is in effect) and an exponent part consisting of *e* digits. (See Section 15.3.6 for a discussion of the scale factor.) The *e* has no effect on input, and *d* has no effect on input if the input field contains a decimal point. Neither the value of *d* nor the value of *e* may exceed the value of *w*. Examples of the effects of `E` and `D` editing are given in Table 15, page 269.

Copyright © 1989–1994 Thinking Machines Corporation

Input: E*w*.*d*, D*w*.*d*, and E*w*.*d*E*e* edit descriptors

The form and interpretation of the input field is the same as for F editing (15.2.1.3.1).

Output: E*w*.*d* edit descriptor

A real value is output in one of two forms when edited using the E*w*.*d* edit descriptor:

[blanks] [±]$x_1x_2\ldots x_k$.$y_1y_2\ldots y_{d-k+1}$E±z_1z_2 (if $0 < k < d+2$ and $|exp| \le 99$)
[blanks] [±][0].$0_10_2\ldots0_{|k|}y_1y_2\ldots y_{d-|k|}$E±$z_1z_2$ (if $-d < k \le 0$ and $|exp| \le 99$)

where the x_i, y_i, and z_i represent decimal digits. The scale factor *k* effectively shifts the decimal point to the right or left. The value of *k* must be in the range $-d < k < d+2$.

The output forms below are used if the exponent is 100 or greater (optional leading blanks are not indicated):

[±]$x_1x_2\ldots x_k$.$y_1y_2\ldots y_{d-k+1}$±$z_1z_2z_3$ (if $0 < k < d+2$ and $99 < |exp| \le 999$)
[±][0].$0_10_2\ldots0_{|k|}y_1y_2\ldots y_{d-|k|}$±$z_1z_2z_3$ (if $-d < k \le 0$ and $99 < |exp| \le 999$)

The representation is right-justified in the field and may contain leading blanks. The field contains a leading minus if the value of the edited data item is negative, or an optional leading plus if the value is positive (a leading plus may be forced using the SP edit descriptor).

The magnitude and fractional part of the value are represented as unsigned strings of decimal digits separated by a decimal point (either string may be empty). The string of digits represents the magnitude of the internal value as modified by the established scale factor and rounded to *d* fractional digits. Leading zeros are not output except for an optional zero immediately to the left of the decimal point if the magnitude of the value in the output field is less than one. A zero appears as the magnitude if there would otherwise be no digits in the output field.

The digits z_i of the exponent part represent the power of ten by which the significand part is to be multiplied to obtain the value represented.

Output: D*w*.*d* edit descriptor

A real value is output in a form identical to that produced when using the E*w*.*d* edit descriptor, except that a D replaces the E in the exponent).

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Output: `E`*w*`.`*d*`E`*e* edit descriptor

A real value is output in one of the following forms when edited using the `E`*w*`.`*d*`E`*e* edit descriptor (optional leading blanks are not indicated):

$[\pm]x_1x_2\ldots x_k.y_1y_2\ldots y_{d-k+1}\pm z_1z_2\ldots z_e$ (if $0 < k < d+2$ and $|exp| \le 10^e - 1$)
$[\pm].0_1 0_2\ldots 0_{|k|}y_1y_2\ldots y_{d-|k|}\pm z_1z_2\ldots z_e$ (if $-d < k \le 0$ and $|exp| \le 10^e - 1$)

The x_i, y_i, and z_i represent decimal digits. The scale factor k effectively shifts the decimal point to the right or left; the value of k must be in the range $-d < k < d+2$. The sign in the exponent part is always produced; a plus sign is produced if the exponent value is zero or positive.

The representation of the magnitude of the value is identical to that described for the `E`*w*`.`*d* edit descriptor.

The digits $z_1z_2\ldots z_e$ of the exponent part represent the power of ten by which the significand part is to be multiplied to obtain the value represented.

Table 15. Effects of I/O editing using `E` and `D` edit descriptors.

Data edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 1234567890123456789012
E7.1	123	12.3000	0.1E+02
	.123	0.123000	0.1E+00
	123.	123.000	0.1E+03
D10.5	3.14159	3.14159	.31416D+01
	5e10	500000.	.50000D+06
	5d10	500000.	.50000D+06
	6.022 +24	6.022000E+24	.60220D+25
E20.12	3.1415926535897	3.14159	0.314159265359E+01
	1d12	1.00000	0.100000000000E+01
	4444444e12	4.444444E+06	0.444444400000E+07
	-2.28	-2.28000	-0.228000000000E+01
E20.12E3	3.1415926535897	3.14159	0.314159265359E+001
	4444444e12	4.444444E+06	0.444444400000E+007

15.2.1.3.3 Complex Editing. A complex data item consists of a pair of separate real data items; accordingly, editing is specified by two edit descriptors, each of which specifies the editing of real data. The first of the edit descriptors specifies the real

Copyright © 1989–1994 Thinking Machines Corporation

part; the second specifies the imaginary part. The two edit descriptors may be different. Control edit descriptors and character literal edit descriptors may intervene between the edit descriptors for the real and the imaginary parts.

15.2.1.4 Generalized Numeric Editing (G Editing)

The G*w*.*d* and G*w*.*d*E*e* edit descriptors are used with an I/O list item of type real or complex. These edit descriptors indicate that a field to be edited occupies *w* positions, the fractional part of which consists of a maximum of *d* digits and the exponent part of which consists of *e* digits. Neither the value of *d* nor the value of *e* may exceed the value of *w*. Examples of the effects of G editing are given in Table 16.

Table 16. Effects of I/O editing using G edit descriptors.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 123456789012345678901
G20.6	123456789	123.457	123.457
	123456789012345	1.234568E+08	0.123457E+09
	1.23	1.23000	1.23000
	54321	5.432100E-02	0.543210E-01
	1+5	0.100000	0.100000
G9.3	1000.	1000.00	0.100E+04
	100.	100.000	100.
	10.	10.0000	10.0
	1.	1.00000	1.00
	0.	0.000000E+00	0.000E+00
	.1	0.100000	0.100
	.01	1.000000E-02	0.100E-01
	.001	1.000000E-03	0.100E-02
	-1.	-1.00000	-1.00
	-12.	-12.0000	-12.0
	-123.	-123.000	-123.
	-1234.	-1234.00	-.123E+04
G20.5E3	123456789012345	1.234568E+09	0.12346E+010
	6.022+ 23	6.022000E+23	0.60220E+024
G20.5E4	123456789012345	1.234568E+09	0.12346E+0010
	6.022d23	6.022000E+23	0.60220E+0024

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Input: G*w*.*d* and G*w*.*d*E*e* edit descriptors

The form and interpretation of input fields for G editing is the same as for F editing (Section 15.2.1.3.1).

Output: G*w*.*d* and G*w*.*d*E*e* edit descriptors

The editing of output fields using the G edit descriptor is defined in terms of the F and E edit descriptors. The method of representation of an output field depends on the magnitude N of the data item being edited, as indicated in Table 17. The trailing blanks in the representation place the significand portion of the output at the same relative position in the field regardless of whether E or F editing is used. Note that the scale factor *k* has no effect if the magnitude of the data item to be edited is within a range that permits effective use of F editing.

Table 17. Dependence of G output representation on magnitude of data item.

Magnitude of Item	Equivalent Conversion	
	G*w*.*d*	G*w*.*d*E*e*
$0 \le N < 10^{-1}$	*k*P,E*w*.*d*	*k*P,E*w*.*d*E*e*
$10^{-1} \le N < 10^{0}$	F(*w*–4).*d*, 4(" ")	F(*w*–*n*).*d*, *n*(" ")
$10^{0} \le N < 10^{1}$	F(*w*–4).(*d*–1), 4(" ")	F(*w*–*n*).(*d*–1),*n*(" ")
$10^{1} \le N < 10^{2}$	F(*w*–4).(*d*–2), 4(" ")	F(*w*–*n*).(*d*–2),*n*(" ")
. . .	. . .	. . .
$10^{d-2} \le N < 10^{d-1}$	F(*w*–4).1, 4(" ")	F(*w*–*n*).1, *n*(" ")
$10^{d-1} \le N < 10^{d}$	F(*w*–4).0, 4(" ")	F(*w*–*n*).0, *n*(" ")
$10^{d} \le N$	*k*P,E*w*.*d*	*k*P,E*w*.*d*E*e*

NOTE: *k* is the scale factor currently in effect, and *n* is *e*+2

15.2.2 Logical Editing (L Editing)

The L*w* edit descriptor indicates that a field to be edited occupies *w* positions. The corresponding I/O list item must be of type logical. Examples of the effects of L editing are given in Table 18, page 272.

Copyright © 1989–1994 Thinking Machines Corporation

Input: L*w* edit descriptor

A logical field to be edited must have the form:

[blanks][.] T *[chars]*
[blanks][.] F *[chars]*

Leading blanks and a period in the field are ignored. The first nonblank alphabetic character must be a T representing true or an F representing false. Additional characters following the T or F are ignored. Note that the literal constant forms .TRUE. and .FALSE. are acceptable input forms.

Output: L*w* edit descriptor

A logical value is output in one of the following forms:

[blanks] T
[blanks] F

The output field consists of a T or F, depending on whether the value of the data item is true or false, respectively. The representation is right-justified in the field, preceded by *w*−1 leading blanks.

Table 18. Effects of I/O editing using L edit descriptors.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 12345678901234567890
L6	.true.	.TRUE.	T
	.false.	.FALSE.	F
	t	.TRUE.	T
	f	.FALSE.	F
L8	true!	.TRUE.	T
	false!	.FALSE.	F
	.T	.TRUE.	T
	.F	.FALSE.	F

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

15.2.3 Character Editing (A Editing)

The A and A*w* edit descriptors are used with an I/O list item of type character. Examples of the effects of A editing are given in Table 19.

Input: A and A*w* edit descriptors

A character field to be edited has the form:

$$c_1 c_2 \ldots c_w$$

where the c_i are ASCII characters. If a field width *w* is specified with the A*w* edit descriptor, the field consists of *w* characters. If a field width *w* is not specified, the number of characters in the field is the length of the character I/O list item. The width is measured in character units, one unit per character.

Let *len* be the length of the associated I/O list character item. If the specified field width *w* for A*w* input is greater than or equal to *len*, the rightmost *len* characters are taken from the input field. If the specified field width *w* is less than *len*, the *w* characters appear left-justified, with *len* – *w* trailing blanks.

Table 19. Effects of format editing using A edit descriptors for character items of length *len*.

len	Data Edit Descriptor	Input Field 123456789012	Internal Value	Output Field 123456789012345
1	A	xyzzy	x	x
4	A	xyzzy	xyzz	xyzz
7	A	xyzzy	xyzzy	xyzzy
10	A	xyzzy	xyzzy	xyzzy
1	A	123456789012	1	1
4	A	123456789012	123456789012	1234
7	A	123456789012	1234567	1234567
10	A	123456789012	1234567890	1234567890
1	A3	123456	3	3
4	A3	123456	123	123
7	A3	123456	123	123
10	A3	123456	123	123
1	A9	1234567890	9	9
4	A9	1234567890	6789	6789
7	A9	1234567890	3456789	3456789
10	A9	1234567890	123456789	123456789

Copyright © 1989–1994 Thinking Machines Corporation

Output: `A` and `A`*w* edit descriptors

A character value is output in the following form:

$c_1 c_2 ... c_w$

Let *len* be the length of the associated I/O list character item. The output field consists of *w*−*len* leading blanks followed by the *len* characters from the internal representation if the specified field width *w* is greater than *len*. The output field consists of the left-most *w* characters from the internal representation if the specified field width *w* is less than or equal to *len*.

15.3 Control Edit Descriptors

A *control edit descriptor* (R15-7) is an edit descriptor that may affect the conversions performed by subsequent data edit descriptors, but does not itself cause the transfer of data or the conversion of data.

A *control-edit-desc* is one of: R15-7

- *position-edit-desc*
- *[r] /*
- `:`
- `$`
- *sign-edit-desc*
- *k* `P`
- *blank-edit-desc*

A *k* is: R15-8

[sign] integer-literal-constant

In *k* `P`, *k* is called the *scale factor*.

The interpretation of control edit descriptors is given in the following sections.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

15.3.1 Position Editing

The T, TL, TR, and X edit descriptors (R15-9) specify the position at which the next character is to be transmitted to or from the record.

A *position-edit-desc* is one of: R15-9

- T *n*
- TL *n*
- TR *n*
- *n* X

An *n* is R15-10
integer-literal-constant

- *n* must be positive.

The position specified by a T edit descriptor may be in either direction from the current position. On input, this allows portions of a record to be processed more than once, possibly with different editing.

The position specified by an X edit descriptor is forward from the current position. On input, a position beyond the last character of the record may be specified if no characters are transmitted from such positions. An *n*X edit descriptor has the same effect as a TR*n* edit descriptor.

On output, a T, TL, TR, or X edit descriptor does not by itself cause characters to be transmitted and therefore does not by itself 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. The result is as if the entire record were initially filled with blanks.

On output, a character in a record may be replaced. However, a T, TL, TR, or X edit descriptor never directly causes a character already placed in the record to be replaced, but may result in positioning such that subsequent editing causes a replacement.

15.3.1.1 T, TL, and TR Editing

The T*n* edit descriptor indicates that the transmission of the next character to or from a record is to occur at the *n*th character position of the record.

The TL*n* edit descriptor indicates that the transmission of the next character to or from the record is to occur at the character position *n* characters backward from the current position. However, if the value of *n* would place the current position before position one, the transmission of the next character to or from the current record occurs at position one.

Copyright © 1989–1994 Thinking Machines Corporation

The TR*n* edit descriptor indicates that the transmission of the next character to or from the record is to occur at the character position *n* characters forward from the current position.

The following READ statement reads 6 characters as a character string, then *rereads* the same 6 characters and converts them to an integer:

```
      CHARACTER*6 STR
      READ( *, '( A, T1, I6 )' ) STR, I
```

An example of output editing using the T*n*, TL*n*, and TR*n* edit descriptors:

```
      PRINT 10, 100, 200, 300
      PRINT 20, 200, 300, 100
      PRINT 30, 300, 100, 200
   10 FORMAT( T21, I5, T11, I5, T1, I5 )
   20 FORMAT( T11, I5, TL15, I5, TR15, I5 )
   30 FORMAT( T1, I5, TR15, I5, TL15, I5 )
```

All three print statements yield the same line of output:

```
  300       200       100
```

15.3.1.2 X Editing

The *n*X edit descriptor indicates that the transmission of the next character to or from a record is to occur at the position *n* characters forward from the current position. Note that the *n* must be specified and must be positive. Examples of the effects of X editing are given in Table 20, page 276.

Table 20. Effects of format editing using X edit descriptors.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 123456789012345678901
3X,I3	no −23	−23	−23
	12345678	456	456
2X,E12.4	123456789	345.679	0.3457E+03
	zz−23	−2.300000E−03	−.2300E−02

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

15.3.2 Slash Editing

The slash edit descriptor indicates the end of data transfer on the current record.

On input from a file connected for sequential access, the remaining portion of the current record is skipped and the file is positioned at the beginning of the next record, which becomes the current record. On output to a file connected for sequential access, a new empty record is created following the current record; this new record then becomes the last and current record of the file and the file is positioned at the beginning of this new record.

For a file connected for direct access, the record number is increased by one and the file is positioned at the beginning of the record that has that record number, and this record becomes the current record.

Note that a record containing no characters may be written. If the file is an internal file or a file connected for direct access, the record is filled with blank characters. Note also that an entire record may be skipped on input. A repeat specification prefixing the slash edit descriptor is optional. If it is not specified, the default value is one.

NOTE: The comma used to separate each *edit-descriptor* in an *edit-descriptor-list* (R15-2) may be omitted before a slash edit descriptor when the optional repeat specification *r* is not present, and always after a slash edit descriptor.

The following example illustrates the use of slash editing of a file connected for sequential access:

```
      INTEGER II(6)
      WRITE( 10, 100 ) 101, 102, 103, 104, 105, 106, 107, 108,
     $  109, 110, 111, 112, 113
100   FORMAT( 4I5 / 2I5 / 5I5 )
      REWIND( 10 )
      READ( 10, 200 ) II
200   FORMAT( I5 / 2I5 )
      PRINT 200, II
      END
```

The slash edit descriptors in the first `FORMAT` statement cause the 13 output items to be written as four records:

```
101  102  103  104
105  106
107  108  109  110  111
112  113
```


Copyright © 1989–1994 Thinking Machines Corporation

The fourth record is begun because format reversion (Section 15.5.1) has the same effect as a slash edit descriptor. When the records are read back using the second FORMAT statement, the following values are read:

```
101
105  106
107
112    113
```

The slash edit descriptor causes parts of records 1 and 3 to be skipped.

15.3.3 Colon Editing

The colon edit descriptor terminates format control if there are no more items to be processed in the I/O list, otherwise it has no effect.

NOTE: The comma used to separate each *edit-descriptor* in an *edit-descriptor-list* (R15-2) may be omitted before or after a colon edit descriptor.

The following code fragment illustrates the use of colon edit descriptors:

```
        PRINT 10, 14, 12.1, 12.2
        PRINT 10, 15, 13.2
        PRINT 10, 16
10      FORMAT( ' Num=', I3 : '  X=', F6.2 : '  Y=', F6.2 )
```

The three PRINT statements produce the following output:

```
Num= 14  X= 12.10  Y= 12.20
Num= 15  X= 13.20
Num= 16
```

Without the colon edit descriptors, the output is:

```
Num= 14  X= 12.10  Y= 12.20
Num= 15  X= 13.20  Y=
Num= 16  X=
```

15.3.4 Dollar Editing

The dollar edit descriptor suppresses the line break that normally occurs at the end of an output record. The dollar edit descriptor does not terminate format control, and can appear anywhere in the edit descriptor list of a format specification. The dollar edit descriptor is

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

not valid for editing input records, but is useful for displaying a prompt message in interactive programs that prompt for user input.

NOTE: The comma used to separate each *edit-descriptor* in an *edit-descriptor-list* (R15-2) may be omitted before or after a dollar edit descriptor.

The following code fragment illustrates the use of the dollar edit descriptor:

```
      PRINT 10
10    FORMAT( " Please enter your name: ", $ )
      READ 20, LINE
20    FORMAT( A40 )
      PRINT *, "Hello " // LINE
```

An interactive session using this fragment might look like:

```
Please enter your name: Fred Flintstone
Hello Fred Flintstone
```

15.3.5 S, SP, and SS Editing

The sign edit descriptors `S`, `SP`, and `SS` are used to control optional plus characters in numeric output fields (but not in integer fields formatted with the `O` or `Z` edit descriptors). Examples of the effects of sign editing are given in Table 21, page 280.

A *sign-edit-desc* is one of: R15-11

- `S`
- `SP`
- `SS`

The interpretation of an `SP` edit descriptor causes a plus sign to be *produced* in any subsequent output field position that normally contains an optional plus sign, while interpretation of an SS or `S` edit descriptor causes optional plus signs to be *suppressed*. At the beginning of execution of each formatted output statement, optional plus signs are suppressed, as if an `SS` or `S` edit descriptor had been encountered.

The `S`, `SP`, and `SS` edit descriptors affect only `I`, `F`, `E`, `D`, and `G` editing during the execution of an output statement, and have no effect during the execution of an input statement.

Copyright © 1989–1994 Thinking Machines Corporation

Table 21. Effects of format editing using the sign edit descriptors.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 123456789012345678901
S,I3	-23	-23	-23
	+23	23	23
SS,I3	-23	-23	-23
	23	23	23
SP,I3	-23	-23	-23
	23	23	+23
S,E10.4	123456789	12345.7	0.1235E+05
	-123456789	-12345.7	-.1235E+05
SS,E10.4	123456789	12345.7	0.1235E+05
	-123456789	-12345.7	-.1235E+05
SP,E10.4	123456789	12345.7	+.1235E+05
	-123456789	-12345.7	-.1235E+05

15.3.6 P Editing and the Scale Factor

The *k*P edit descriptor sets the value of the scale factor to *k*. The scale factor may affect the editing of numeric quantities. Examples of the effects of scale factor on numeric output editing are given in Table 22.

The value of the scale factor is zero at the beginning of execution of each input/output statement. It applies to all subsequent F, E, D, and G edit descriptors until another P edit descriptor is encountered, which establishes a new scale factor. Note that reversion of format control (Section 15.5.1) does not affect the established scale factor.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Table 22. Effects of scale factor on I/O editing.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 123456789012345678901
3P,F12.4	1	1.000000E-07	0.0001
	1.	1.000000E-03	1.0000
	1e1	1.000000E-03	1.0000
	1.e1	10.0000	10000.0000
-2P,F12.4	1	1.000000E-02	0.0001
	1.	100.000	1.0000
	1e5	10.0000	0.1000
	1.e5	100000.	1000.0000
-3P,F20.3	12345	12345.0	12.345
	123456	123456.	12345.678
	12345678	1.234568E+07	12345.678
	.123	123.000	0.123
4P,F20.0	12345678	1234.57	12345678.
	1.2345678	1.234568E-04	1.
	123456e-4	12.3456	123456.
3PG20.6	123	1.230000E-07	123.0000E-09
	123456	1.234560E-04	123.4560E-06
	.000123456	1.234560E-07	123.4560E-09
-1PG15.4	12345	12.3450	12.34
	12345e0	1.23450	1.234
	1e5	10.0000	10.00

The scale factor *k* affects editing in the following manner:

- On input with `F`, `E`, `D`, and `G` editing (provided that no exponent exists in the field), the effect of the scale factor is that the externally represented number is *divided* by 10^k prior to conversion to internal form.
- On input with `F`, `E`, `D`, and `G` editing, the scale factor has no effect if there is an exponent in the field.
- On output with `F` editing, the effect of the scale factor is that the internally represented number is *multiplied* by 10^k prior to output.
- On output with `E` and `D` editing, the significand part of the quantity to be produced is multiplied by 10^k and the exponent is reduced by *k* (resulting in no net change in the apparent magnitude of the number).

Copyright © 1989–1994 Thinking Machines Corporation

- On output with G editing, the scale factor has an effect only if the magnitude of the value to be output is in a range that permits the use of F editing. If the magnitude is such that the use of E editing is required, the scale factor has the same effect as with E output editing.

NOTE: The comma used to separate each *edit-descriptor* in an *edit-descriptor-list* (R15-2) may be omitted between a P edit descriptor and a subsequent F, E, D, or G edit descriptor.

15.3.7 BN and BZ Editing

The BN and BZ edit descriptors may be used to specify the interpretation of blanks embedded in numeric input fields. Examples of the effects of blank interpretation on numeric input editing are given in Table 23.

A *blank-edit-desc* is one of: R15-12

- BN
- BZ

At the beginning of execution of each formatted input statement, nonleading blank characters read from a file connected by an OPEN statement are interpreted as zeros or are ignored, depending on the value of the BLANK= specifier (Section 14.2.6) currently in effect for the unit; a preconnected file or an internal file is treated as if the file had been opened with BLANK=NULL. If a BN edit descriptor is encountered in a format specification, all nonleading blank characters in succeeding numeric input fields are ignored. The effect of ignoring blanks is to treat the input field as if blanks had been removed, the remaining portion of the field right-justified, and the blanks replaced as leading blanks. However, a field containing only blanks has the value zero. If a BZ edit descriptor is encountered in a format specification, all nonleading blank characters in succeeding numeric input fields are treated as zeros.

The BN and BZ edit descriptors affect only I, F, E, D, and G editing during execution of an input statement. They have no effect during execution of an output statement.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Table 23. Effects of blank interpretation on I/O editing.

Data Edit Descriptor	Input Field 123456789012345	Internal Value	Output Field 123456789012345678901
BN,I8	12 34	1234	1234
	1 2 3 4	1234	1234
BZ,I8	-12 34	-12034	-12034
	-1 2 3 4	-1020304	-1020304
BN,F15.4	12 34	0.123400	0.1234
	1 2 3 4	0.123400	0.1234
	1 +1	1.000000E-03	0.0010
BZ,F15.4	12 34	1.20340	1.2034
	1 2 3 4	102.030	102.0304
	1 +1	10.0000	10.0000

15.4 Character Literal Edit Descriptors

A *character literal edit descriptor* (R15-13) is an edit descriptor used only for output editing. The character literal edit descriptors provide constant data to be output, and are not valid for input.

A *char-literal-edit-desc* is one of: R15-13
- *char-literal-constant*
- *n*H *char [char]...*

An *n* is an: R15-14
integer-literal-constant

- *n* must be positive.
- The *char* in the *n*H form must be of character type.

Each *char* in a character literal edit descriptor must be an ASCII character. In the *n*H edit descriptor, *n* specifies the number of characters following the `H` that comprise the descriptor. The case of characters is significant.

Copyright © 1989–1994 Thinking Machines Corporation

15.4.1 Character Literal Constants (Apostrophe Editing)

The *character literal constant* edit descriptor causes the delimited characters, including blanks, to be written to an output field. Note that a delimiter is either an apostrophe or a quote.

For a character constant edit descriptor, the width of the output field is the number of characters enclosed by the delimiting characters. Within the field, two consecutive delimiting characters are counted as a single character.

The following example illustrates the use of character literal constant edit descriptors:

```
      PRINT 100, 7, 4, 76
100   FORMAT( ' Person''s DOB: ', I2.2, 2( '/', I2.2 ) )
```

The result is:

```
Person's DOB: 07/04/76
```

15.4.2 Hollerith Constants (H Editing)

The *n*H edit descriptor causes character information to be written from the next *n* characters (including blanks) following the H of the *n*H edit descriptor.

The following example illustrates the use of Hollerith constant edit descriptors:

```
      PRINT 10, 7, 4, 76
10    FORMAT( 15H Person's DOB: , I2.2, 2( 1H/, I2.2 ) )
```

The statements yield the same result as the example of the previous section:

```
Person's DOB: 07/04/76
```

Note that the apostrophe (') is not doubled, and the slash (/) has no special significance within the Hollerith edit descriptor.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Table 24. Summary of edit descriptors of a format specification.

Edit Descriptor	Description
[r] I *w [. m]*	INTEGER data field
[r] B *w [. m]*	INTEGER data field (binary representation)
[r] O *w [. m]*	INTEGER data field (octal representation)
[r] Z *w [. m]*	INTEGER data field (hexadecimal representation)
[r] F *w . d*	Real or complex data field
[r] E *w . d [* E *e]*	Real or complex data field
[r] D *w . d*	Real or complex data field
[r] G *w . d [* E *e]*	Real or complex data field
[r] L *w*	LOGICAL data field
[r] A *[w]*	CHARACTER data field
[r] /	Terminate data transfer of current record
:	Terminate format control if no more I/O list items
k P	Establish a scale factor of *k*
T *n*	Tabulate to position *n*
TL *n*	Tabulate *n* characters left of current position
TR *n*	Tabulate *n* characters right of current position
n X	On input: skip a field of width *n* On output: fill field with *n* blanks
$	Suppress line break in output record
S	Restore optional display of leading "+"
SP	Produce leading "+" on output
SS	Suppress leading "+" on output
BN	Ignore blanks on input
BZ	Regard blanks as zeros on input
"characters"	Literal data (output only)
*c*H *char [char]...*	Literal data, Hollerith form (output only)
[r] (*edit-descriptor-list*)	Regard edit descriptors as a repeatable group

Copyright © 1989–1994 Thinking Machines Corporation

15.5 Interaction between I/O List and Format Specification

Execution of a formatted file I/O statement initiates *format control*, a series of actions that cause successive items of an input/output list to be edited according to instructions in the associated format specification. Each action of format control depends on information provided by:

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

If an I/O list specifies at least one list item, then at least one data edit descriptor must exist in the format specification. Note that an empty format specification of the form ( ) may be used only if the I/O list is empty or if each item is of zero size; in this case, one input record is skipped or one output record containing no characters is written.

A format specification is interpreted from left to right. An edit descriptor beginning with a repeat specification is processed as a list of *r* items, each identical to the edit descriptor without the repeat specification, and separated by commas. Note that an omitted repeat specification is treated in the same way as a repeat specification whose value is one.

Each data edit descriptor in a format specification is associated with a single I/O list item of the statement, except that two `F`, `E`, `D`, or `G` edit descriptors must be interpreted for each item of type complex. Control edit descriptors are not associated with an I/O list item, but may affect the subsequent interpretation of data edit descriptors. Character literal edit descriptors also are not associated with an I/O list item, but are transmitted directly to a record.

Whenever format control encounters a data edit descriptor in a format specification, it determines whether there is a corresponding item specified by 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 (with the next item, if any, and the next data edit descriptor). If there is no such item, format control terminates.

If format control encounters a colon edit descriptor in a format specification and another I/O list item is not specified, format control terminates.

15.5.1 Format Control Reversion

If format control encounters the right-most parenthesis of a complete format specification and another I/O list item is not specified, format control terminates. However, if another I/O list item is specified, the file is positioned at the beginning of the next record, as if a slash edit descriptor had been processed (Section 15.3.2). Format control then *reverts* to the beginning of the edit descriptor list terminated by the last preceding right parenthesis.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

If there is no such preceding right parenthesis, format control reverts to the first edit descriptor of the complete format specification.

If format control reversion occurs, the reused portion of the format specification must contain at least one data edit descriptor. If format control reverts to a parenthesis that is prefixed by a repeat specification, the repeat specification is reused as well. Reversion of format control itself has no effect on the scale factor (Section 15.3.6), the sign control edit descriptors (Section 15.3), or the blank interpretation edit descriptors (Section 15.3.7).

As an example of format control reversion, the two `FORMAT` statements

```
10    FORMAT( L2, 2( F5.1, I3 ), SP, I3 )
10    FORMAT( L2, F5.1, I3, F5.1, I3, SP, I3 / F5.1, I3, F5.1 )
```

have the same effect when used with the `PRINT` statement

```
      PRINT 10, .TRUE., 1.1, 1, 1.2, 2, 3, 2.1, 4, 2.2
```

Both produce the output:

```
T  1.1  1  1.2  2 +3
+2.1 +4 +2.2
```

Format control reverts in the first `FORMAT` statement after printing the value `+3` using the last `I5` edit descriptor in the format specification. The second line of output begins the next record and is the result of the (implied) slash edit descriptor that results from reversion. Note that leading pluses continue to be produced after format reversion.

15.6 List-Directed Formatting

The use of an asterisk (`*`) as a `FMT=` I/O control specifier in a file I/O statement causes *list-directed* editing to be performed by the statement (see Section 13.4.1.2). The characters in one or more list-directed records comprise a sequence of variable-length fields separated by field separators. Each *variable-length field* is one of:

- a *constant field*, with one of the following forms:
  - a *numeric field* — an integer, real, or complex field, as described below. On input, a numeric constant field can be assigned to a corresponding input list variable of any numeric type.
  - a *logical field* — a logical constant representing true or false.

Copyright © 1989–1994 Thinking Machines Corporation

  - a *character field*. On input, character fields must be delimited by apostrophes or quotes, and the delimiting character must be doubled (two contiguous appearances of the character represent a single character). On output, character fields are not delimited, and the delimiting character is not doubled.

- a *repeated field*, of the form *r*`*`*c*, where *r* is a positive integer literal constant and *c* is one of the constant fields described above (blanks may not separate the `*` from the *r* or the *c*). The form *r*`*`*c* is equivalent to *r* successive appearances of the constant field *c*.
- a *null field*, consisting of one or more blanks.
- a *repeated null field*, of the form *r*`*`, where *r* is a positive integer literal constant (blanks may not separate the *r* from the `*`). The form *r*`*` is equivalent to *r* successive appearances of the null field.

A *field separator* in a record can be any one of the following:

- one or more blanks — this form of field separator may only be used to separate two nonblank fields
- a comma, with or without surrounding blanks or tabs
- a slash, with or without surrounding blanks or tabs

The fields may cross record boundaries subject to restrictions described below. The end of a record has the same effect as a blank character unless it is within a character field. Any sequence of two or more consecutive blanks is treated as a single blank, except within a character field.

15.6.1 List-Directed Input

The form that an input field may take depends on the type of the corresponding item in the list. Input forms that are acceptable when performing formatted input of an I/O list item are also acceptable for list-directed input of an item of that type.

Input: Integer list items

When the corresponding input item is of type integer, the input form is that of an integer input field that is suitable for `I` editing (Section 15.2.1.1).

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

Input: Real list items

When the corresponding input item is of type real, the input form is that of a numeric input field. A numeric input field is suitable for `F` editing (Section 15.2.1.3.1), and the field is assumed to have no fractional digits unless a decimal point appears within the field.

Input: Complex list items

When the corresponding item is of type complex, the input form consists of a left parenthesis followed by a pair of numeric input fields separated by a comma, followed by a right parenthesis. The first numeric input field is the real part and the second is the imaginary part of the complex field. The end of a record may occur between the real part and the comma or between the comma and the imaginary part.

Input: Logical list items

When the corresponding item is of type `LOGICAL`, the input form is that of a logical input field suitable for `L` editing (Section 15.2.2), except that the field must not include slashes, blanks, or commas among the trailing characters permitted when using `L` editing.

Input: Character list items

When the corresponding item is of type character, the input form consists of a character literal constant (R4-19, page 29). Character constants may cross record boundaries, but the end of a record must not occur between a doubled apostrophe in an apostrophe-delimited constant, nor between a doubled quote in a quote-delimited constant. The end of the record does not cause a blank or any other character to become part of the constant. The constant may be continued on as many records as needed. The characters blank, comma, and slash may appear in character constants.

Let *len* be the length of the corresponding input list item, and let *w* be the length of the character constant. If *len* is less than or equal to *w*, the leftmost *len* characters of the constant are transmitted to the corresponding input list item. If *len* is greater than *w*, the constant is transmitted to the leftmost *w* characters of the corresponding item and the remaining *len*−*w* characters of the corresponding item are filled with blanks. The effect is as though the constant field were assigned to the corresponding item in a character assignment statement (see Section 10.1).

NOTE: Any numeric input field format is acceptable for use as input to a numeric input list item. A numeric value of one type is converted to a value of another type according to the rules of assignment conformance (Section 10.1.1).

Copyright © 1989–1994 Thinking Machines Corporation

Blanks are never interpreted as zeros during list-directed input, even if the BLANK=ZERO specifier is used in the open statement. Blanks in a list-directed input record are considered to be part of a field separator, except for the following contexts:

- blanks embedded in 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.

15.6.1.1 Null Fields

A null field indicates that the value of a corresponding input list item is to remain unchanged. When the list item corresponding to a null input field is of type complex, the effect is that the entire list item remains unchanged (not just the real part or imaginary part).

A null field is specified by having no characters between successive field separators, or no characters preceding the first nonblank field separator in the first record read by each execution of a list-directed input statement, or by the *r** form. Note that the end of a record following any non-null field separator, with or without intervening blanks, does not specify a null field.

A slash encountered as a field separator during list-directed input causes termination of the input statement after the assignment of the previous field. If there are additional items in the input list, the effect is as if null fields had been supplied for them (that is, their values remain unchanged). Any index variable in an implied DO of the input list is defined as though null fields had been supplied for any input list items for which there is no corresponding input field. (It is therefore not possible to tell from the value of the index variable how many null fields were read.)

An example involving list-directed input is:

```
INTEGER I
REAL X(8)
CHARACTER*11 P
COMPLEX Z
LOGICAL G
...
READ *, I, X, Z, P, G
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

If the input data records are:

```
12345 6789,, 2*1.5   4*
'ISN''T
 BOB''S', ( 123,
0 ), .TRUE.
```

the results are:

Variable	Value
I	12345
X(1)	6789.0
X(2)	(unchanged)
X(3)	1.5
X(4)	1.5
X(5) – X(8)	(unchanged)
P	ISN'T BOB'S
Z	(123.0,456.0)
G	.TRUE.

The null input fields in the first input data record cause some of the elements of the array X to remain unchanged.

15.6.2 List-Directed Output

The form of the fields produced during list-directed output is the same as for input, with the exception of character fields. The following general rules apply to list-directed output:

- Each output record begins with a blank character to provide carriage control.
- Individual fields other than character fields are separated by one or more blanks; adjacent character fields are written with no intervening blanks.
- No output field is split over a record boundary, with the exception of character fields longer than one record length and complex constants, which may be split after the comma.
- List-directed output statements do not produce null fields, slash field separators, or repeated forms constant fields.

Copyright © 1989–1994 Thinking Machines Corporation

Output: Logical list items

Logical output constants are `T` for the value true and `F` for the value false, produced using the effect of an `L2` edit descriptor.

Output: Integer list items

Integer values are produced with the effect of an `I12` edit descriptor.

Output: Real list items

Single-precision real values are produced with the effect of a `1PG15.7` edit descriptor list.

Double-precision real values are produced with the effect of a `1PG24.16` edit descriptor list.

Output: Complex list items

Single-precision complex values are produced with the effect of a `"(", 1PG14.7, 1PG14.7 ")"` edit descriptor list.

Double-precision complex values are produced with the effect of a `"(", 1PG23.16, 1PG23.16, ")"` edit descriptor list.

The end of a record will occur between the comma and the imaginary part only if the output field is as long as, or longer than, an entire record. The only embedded blanks occurring within a complex constant are between the comma and the end of a record and one blank at the beginning of the next record.

Output: Character list items

Character values produced for a file opened normally are produced with the effect of an `A`*n* edit descriptor, where *n* is the length of the character expression. Character fields produced for a file opened normally:

- are not delimited by apostrophes or quotation marks,
- are not preceded or followed by a field separator,
- have each internal apostrophe or quotation mark represented externally by only one apostrophe or quotation mark, and

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

- have a blank character inserted for carriage control at the beginning of any record that begins with the continuation of a character constant from the preceding record.

The following example program illustrates list-directed output for all the different data types.

```
      INTEGER HUNDREDS(4)
      DATA HUNDREDS / 100, 200, 300, 400 /
      CHARACTER HEXCHAR(6)
      DATA HEXCHAR / 'A', 'B', 'C', 'D', 'E', 'F' /
      PRINT *, .TRUE., .FALSE., HUNDREDS, HEXCHAR,
     $  5280*12,
     $  3.1415926535897926,   6.022e23,
     $  3.1415926535897926d0, 6.022d23,
     $  ( 1, 1 ), ( 2.0, 2.0 ), ( 3, 3.0 ), ( 4, 4 ),
     $  ( 5.0d0, 5.0d0 ), ( 6, 6.0d0 ), ( 7.0d0, 7 ),
     $  ( 1, 2 ) * ( 3, 4 ),
     $  'Tinker', 'Tailor', 'Soldier', 'Poor man ',
     $  'The quick brown fox jumped over the lazy dog''s
back',
     $  'Once upon a midnight dreary...', 'Finnegan''s awake'
      END
```

The program writes eight list-directed records to the standard output file (each line represents a record):

```
T F         100         200         300         400ABCDEF       63360
  3.141593      6.0220002E+23   3.141592653589793      6.0220000000000000E+23
(1.000000,1.000000) (2.000000,2.000000) (3.000000,3.000000)
(4.000000,4.000000)
(5.000000000000000,5.000000000000000) (6.000000000000000,6.000000000000000)
(7.000000000000000,7.000000000000000) (-5.000000,10.00000)TinkerTailorSoldier
Poor man The quick brown fox jumped over the lazy dog's back
Once upon a midnight dreary...Finnegan's awake
```

15.7 Namelist Editing

The NML= I/O control specifier, as described in Section 13.4.1.3, can be used with READ and WRITE statements to perform input and output to a namelist. A namelist is a set of variables grouped together by the NAMELIST statement (R5-62). The NAMELIST statement is described in Section 5.7.

Copyright © 1989–1994 Thinking Machines Corporation

This section describes the required format for namelist records to be used with namelist input and output statements.

15.7.1 Namelist Input

A namelist input record specifies a sequence of the variables that appear in the namelist and the values that are to be assigned to them. If variables in the namelist do not appear in the namelist input record, they are not changed.

A namelist input record consists of:

- optional blanks
- an ampersand character (`&`) followed immediately by
- the namelist name (R5-62) as specified in the namelist input statement
- one or more blanks
- a sequence of name-value subsequences separated by field separators
- a slash to terminate the record.

NOTE: An alternative format substitutes a dollar sign (`$`) for the ampersand and terminates the record with `$END` instead of a slash.

Examples of namelist input records are:

```
&NLIST I=12345, X(1)=12345, X(3:4)=2*1.5, Z=(123,0)/
$NLIST I=12345, X(1)=12345, X(3:4)=2*1.5, Z=(123,0)$END
```

A name-value subsequence consists of:

- one of the variable names, or a subobject designator, from the list of *namelist-objects* (R5-62) associated with the namelist name
- an equals sign (=)
- the value to be assigned to the variable.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

The value is a *variable-length field* which can be:

- a *constant field* (*c*) with one of the following forms:
  - a *numeric field* — an integer, real, or complex field, as described below. On input, a numeric constant field can be assigned to a corresponding input list variable of any numeric type.
  - a *logical field* — a logical constant representing true or false.
  - a *character field*. On input, a character field must be delimited by apostrophes or quotes only if it contains blanks, spaces, or slashes; if it begins with a quote or an apostrophe; or if it begins with a number string followed by an asterisk. Otherwise, character fields on input need not be delimited. If included in the string, the delimiting character must be doubled (two contiguous appearances of the character represent a single character). On output, character fields are delimited by apostrophes.
- a *repeated field*, of the form *r*`*`*c*, where *r* is a positive integer literal constant and *c* is one of the constant fields described above (blanks may not separate the `*` from the *r* or the *c*). The form *r*`*`*c* is equivalent to *r* successive appearances of the constant field *c*.
- a *null field*, consisting of 0 or more blanks separated by commas.
- a *repeated null field*, of the form *r*`*`, where *r* is a positive integer literal constant (blanks may not separate the *r* from the `*`). The form *r*`*` is equivalent to *r* successive appearances of the null field.

The form of the constant fields may be any that is suitable for the type of the variable. For detailed information see Section 15.6.1, which describes the formats for list-directed input.

A null field indicates that the variable is not to be changed. Note that variables in the namelist that are not specified in the namelist input record are also left unchanged.

15.7.2 Namelist Output

Namelist output records are of the same form as that required for namelist input. The output record consists of:

- A blank character, except for continuation of delimited character constants, to allow for carriage control when the record is printed.
- An ampersand character (or, alternatively, a dollar sign) followed immediately by the namelist group name of the variable being output.

Copyright © 1989–1994 Thinking Machines Corporation

- A sequence of name-value subsequences. Each name-value subsequence consists of the name of the variable being output, an equals sign, the value of the variable, and a field separator.
- A slash (or `$END`, if a dollar sign was used instead of an ampersand) to terminate the record.

The format of the constants output is the same as the formats for list-directed output fields. These are described in detail in Section 15.6.2. If the variable is a zero-sized array or a zero-length character entity, nothing is placed in the output record. A null value is not produced by namelist formatting.

If two or more successive values in an array in an output record have identical values, the processor has the option of producing a repeated constant of the form *r*c* instead of the sequence of repeated values.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation