First Experimental Comm Run

Chapter 2

ELEMENTS

2.1 Introduction

A program:declaration written in JOVIAL consists, basically, of statements and declarations. The statements specify the computations to be performed with arbitrarily named data. Simple:statements can be grouped together into compound:statements in order to help in specifying the order of computations. Among the declarations are data:declarations and processing:declarations. The data:declarations name and describe the data on which the program is to operate, including inputs, intermediate results, and final results. The processing:declarations generally contain statements and other declarations. They specify computations, but they differ from statements in that the computations must be performed only when the particular processing:declaration is specifically invoked by name. In addition to statements and declarations, there are directives which serve various purposes. They designate externally defined names the compiler is expected to recognize, they control selective compilation of various statements and declarations, and they provide information the compiler needs in order to optimize the object code. The statements, declarations, and directives are composed of symbols, which are the words of the JOVIAL language. These symbols are, in turn, composed of the signs that constitute the JOVIAL alphabet.

.1 The general order in which the elements of a program:declaration are introduced in the preceding paragraph represents the general order in which one looks up definitions when trying to clear up a question. The definitions in this manual are introduced, however, in the opposite order. Such arrangements lead to complaints that one must "read the book backwards." This comment arises from the process of looking up a form in the table of contents, turning then to the late chapter where it is defined in terms of earlier defined forms. These, more elementary, forms are then found, via the table of contents, in an earlier chapter. And so forth. Nevertheless, the document is arranged for the use of a reader rather than for reference. Difficult as this may be for reference use, the opposite arrangement is much more difficult for a reader.

.2 An index-glossary is included which facilitates reference. The index-glossary answers many questions directly. In other cases, it references syntax equations and sections by number.

2.2 Spaces and Spaces

It is important to distinguish between a space, an element of JOVIAL, and a space, an element of our descriptive language. JOVIAL is written using symbols, the words of the language. The symbols are composed of signs, the elements of the JOVIAL alphabet. In general, symbols do not contain spaces. The exceptions are pointed out in Section 2.5.2, with respect to comment, and in Section 2.8.2, with respect to character:constraints. In general, symbols are separated by spaces. Again the exceptions are noted in Section 2.10; however, these exceptions are permissive; i.e., it is always correct to put spaces between symbols.

.1 The following example is wrong:

```
PLXMPY ( 1. 375, -. 75, 5 ., 7.3 : REAL,
IMAG ) ;
```

.2 The following examples are right:

```
a. BEGIN 1, 3, +5, - 7 END

b. SL:PLXMPY(1.375,-.75,5.,7.3:REAL,IMAG);

c. SL : PLXMPY ( 1.375 , - .75 , 5. , 7.3 :
REAL , IMAG ) ;
```

.3 In defining and explaining signs and symbols, any spaces included in the metalanguage formulas are not meant to be included in the definition. The phrase "string of" implies that there are to be no spaces between the elements strung together. Similarly, phrases such as "followed by", "enclosed in", and "separated by", imply that there are to be no spaces between the elements concerned. This is the situation (except where explicitly stated to be different) in this chapter, Chapter 2. In Chapter 3 and beyond, the opposite view is maintained with respect to these phrases.

2.3 Signs, Elements of the JOVIAL Alphabet

(equ197)

```
              letter
sign  ::=     numeral
              mark
```

(equ134)

letter ::= A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

(equ156)

numeral ::= 0
1
2
3
4
5
6
7
8
9

(equ144)

		+	plus:sign
		-	minus:sign
		*	astrerisk
		/	slash
		\	back:slash
		&	ampersand
		>	greater:than:sign
		<	less:than:sign
		=	equals:sign
		@	at:sign
mark	::=	.	decimal:point
		:	colon
		,	comma
		;	semicolon
			space
		(	left:parenthesis, parenthesis
		)	right:parenthesis, parenthesis
		[	left:backet, bracket
		]	right:bracket, bracket
		'	prime
		"	quotation:mark
		$	dollar:sign
		!	exclamation:point

.1 Sign means a letter, a numeral or a mark. Letter means one of the 26 letters of the English alphabet, written in the form of a roman capital. Numeral means one of the ten arabic numerals: 0,1,2,3,4,5,6,7,8 or 9. (The slash through the zero is only for the purpose of distinguishing it from the letter O in definitions and examples of JOVIAL.) Sign, letter, and numeral are defined more formally by means of the syntax equations in the boxes at the head of this section. Mark is most easily defined by the formal means of the syntax equation in the box above. The box above also contains a metalinguistic term associated with each mark; this serves to define these terms.

2.4 Symbols, The Words of JOVIAL

(equ233)

```
                          primitive
                          ideogram
                          name
                          letter:control:variable
          symbol  ::=     abbreviation
                          number
                          constant
                          comment
                          directive:key
                          status
```

.1 The symbols or words of the JOVIAL language are composed of strings of signs, in some cases a single sign. Most symbols do not contain spaces. In fact, spaces serve to separate symbols from one another.

2.5 PRIMITIVE, Ideogram, Directive:Key, Comment

(equ178)

```
                        ABS
                        ALL
                        ALT             NENT
                        AND             NOT
                        BEGIN           NULL
                        FIT             NWDSEN
                        BLOCK           OR
                        BY              OVERLAY
                        BYTE            PROC
                        DEF             PROGRAM
                        DEFINE          REF
                        DIRECT          REMQUO
                        DSIZE           RESERVE
                        ELSE            RETURN
                        END             SHIFT
    primitive  ::=      ENTER           SIG
                        EQV             SIGNED
                        EXIT            SIGNUM
                        FOR             SIZE
                        FORM            STATUS
                        FORMAT          STOP
                        FRAC            SWITCH
                        GOTO            TABLE
                        IF              TEST
                        IN              THEN
                        INT             TYPE
                        ISIZE           UNTIL
                        ITEM            WHILE
                        JOVIAL          XOR
                        LOC             XRAD
                        NAME            ZAP
```

(equ106)

```
                      +
                      -
                      /
                      **
                      \
                      &
                      =
                      ==
                      <
                      >
                      <=
                      >=
ideogram  ::=         <>
                      .
                      ,
                      :
                      ;
                      !
                      "
                      '
                      (
                      )
                      [
                      ]
                      @
                      @@
```

(equ64)

directive:key ::= |COMPOOL
|SKIP
|BEGIN
|END
|TRACE
|COPY
|ABNORMAL
|SETS
|USES
|POINTER
|ORDER
|RECURSIVE
|TIME
|SPACE
|LINKAGE
|INTERFERENCE
|FREQUENCY

(equ32)

comment ::= " character "

(equ25)

character ::= sign
system:dependent:character

.1 Primitives may be considered the key words of the JOVIAL language. They are generally used to give the primary meaning of a statement or declaration, although some are used for second purposes. Ideograms are generally used as arithmetic:operators, as relational:operators, and for purposes such as grouping, separating, and terminating. Directive:keys are used to state the primary meanings of directives. Comments can be used to annotate a program:declaration; explaining to readers (and often the original programmer) what is going on.

.2 Notice that a comment is delimited by quotation:marks. Therefore, spaces are permitted within a comment, but a quotation:mark is not permitted within a comment. Also, a semicolon is not permitted within a comment. The reason for this is to permit some recovery in case a delimiting quotation:mark is left off a comment. If the comment were not then terminated by the next semicolon, the entire remainder of the program:declaration would be turned inside out; the comments being interchanged with the statements and declarations. Even with this rule, failure to terminate a comment can lead to disaster. If an END is swallowed up, the entire program structure can be disarrayed.

.3 The system:dependent:characters that can be included in comments (and other structures) are simply those characters, other than JOVIAL signs, that the particular system and compiler can read and write.

.4 Notice that primitives, ideograms, and directive:keys do not contain spaces. Spaces are significant in a program:declaration; usually in that they separate symbols. Comments, on the other hand, may contain spaces. This permits easier reading and writing of the commentary. The quotation:marks delimiting the comment provide the necessary grouping so that the spaces do not cause trouble.

.PxP[2]=C

.DefSyn[MonoSpace]=M; .DefSyn[Slant]=S; .DefSyn[BoldFace]=B;

2,6 Abbreviation, Letter:Control:Variable, Name

(equ1)

abbreviation ::= letter

(equ135)

letter:control:variable ::= letter

(equ145)

```
                                   letter
name  ::=     letter               numeral
                $                    $
                                     '
```

,1 Abbreviations are specific letters having specific meanings in specific contexts, usually data:declarations, The specific uses are documented later on without, usually, calling the letter an abbreviation,

,2 The letter:control:variable is a special variable having meaning only within a loop:statement and passing out of existence when the loop:statement is not being executed, It is explained more fully in connection with explanation of the loop:statement,

,3 Regardless of the syntax in the box above, a name must not be the same as any primitive, Notice that a name must include at least two signs, The use of the dollar:sign is system dependent, That is, it provides a means whereby a name can be designated to have some special meaning in relation to the system in which the compiler is embedded, Such special meanings are outside the scope of this manual, however, and names containing dollar:signs are considered the same as other names herein, Names do not contain spaces, An embedded space would change a name into two names or other symbols,

2,7 Number, Constant, Status

(equ154)

number ::= numeral

(equ39)

```
                          numeric:constant
constant:formula  ::=     pattern:constant
                          character:constant
```

(equ26)

character:constant ::= count ' character '

(equ47)

count ::= number

(equ157)

```
                          integer:constant
                          fixed:constant
numeric:constant  ::=     floating:constant
                          status:constant
                          qualified:status:constant
```

(equ222)

status:constant ::= V(status)

(equ187)

```
                                             status:list:name
                                             item:name
     qualified:status:constant  ::=  V(      table:name
   :  status  )
                                             procedure:name

   alternate:entrance:name
```

(equ221)

```
                        primitive
     status  ::=        name
                        letter
```

.1 The above definitions are obviously not complete, in that several kinds of constants mentioned in the box are not yet defined. This discussion is mainly concerned with the use of spaces together with numbers, constants, and statuses as symbols.

.2 A number is a string of numerals, without spaces. In some places, a number can stand alone as a constant. In other places, particularly data:declarations, it stands alone as a symbol but is not considered a constant. In yet other places, a number is part of another symbol. A case in point is the character:constant, defined above. The optional count in a character:constant is a number. (In several places, numbers or other constructs are given new names reminiscent of their uses in those places.)

.3 A character:constant is a symbol. If it begins with a count, there must be no spaces between the count and the first prime. Between the primes, the string of characters may include spaces, but these spaces are significant. They represent part of the value represented by the character:constant. (There are restrictions on the characters permitted in a character:constant, discussed in Section 2.8.2). In a status:constant and a qualified:status:constant, the left:parenthesis, the name, the colon, the status, and the right:parenthesis are all symbols. Spaces are permitted between these elements, but not within the name or the status. Space is not pemitted between V and the left:parenthesis. All other constants are symbols, not containing spaces.

2.8 Constants and Values

(equ39)

```
     constant:formula  ::=   numeric:constant
                             pattern:constant
                             character:constant
```

(equ26)

```
     character:constant  ::=  count  '  character  '
```

(equ47)

```
     count  ::=  number
```

.1 Character:constants are the direct means of representing character values to be manipulated by a program. (Character:variables and character:formulas are indirect means.) The characters acceptable as character values are whatever the system will accept from among those given in the body of Figure 2-1. At least the 59 JOVIAL signs must be accepted. Comparison of Figure 2-1 with Section 2 of USAS X3.4-1968, "USA Standard Code for Information Interchange", shows the graphic characters in identical positions in the two tables. Figure 2-1 includes eight additional columns presently under consideration by standardization bodies. The positions of the characters in the table are the only correspondence. This manual does not require that internal representation be in accordance with USAS X3.4-1968. If, however, JOVIAL program:declarations generate messages for transmission to other systems or process messages received from other systems, these messages are required by other directives to conform to USAS X3.4-1968 in their external representation.

(tab1)

Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Column Code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Row																
0			space	0	@	P		p								
1			!	1	A	Q	a	q								
2			"	2	B	R	b	r								
3			#	3	C	S	c	s								
4			$	4	D	T	d	t								
5			%	5	E	U	e	u								
6			&	6	F	V	f	v								
7			'	7	G	W	g	w								
8			(	8	H	X	h	x								
9			)	9	I	Y	i	y								
10			*	:	J	Z	j	z								
11			+	;	K	[	k									
12			,	<	L	\	l									
13			-	=	M	]	m									
14			.	>	N		n									
15			/	?	O		o									

Notes: row 0, column 3: zero
row 1, column 3: one
row 7, column 2: prime, often rendered as a vertical mark in JOVIAL
row 12, column 6: a lowercase letter
row 15, column 4: an uppercase letter

Figure 2-1. Characters

.2 All of the character values indicated in the body of Figure 2-1 can be represented in character:constants (except for system-dependent limitations). Artifices are required, however, to represent some of the values. Any spaces within the delimiting primes, except within a three-character code, represent characters of value "space". Primes, semicolons, and dollar:signs have special meanings. Therefore, in order to represent a single occurrence of one of these signs, two of them are used in succession. If a succession of these signs is desired as part of the value represented by a character:constant, the entire string is doubled. In summary:

2n primes are used to represent n primes.

2n semicolons are used to represent n semicolons.

2n dollar:signs are used to represent n dollar:signs.

.3 The reason for doubling the primes inside a character:constant is that single prime terminates the constant. The reason for doubling semicolons inside a character:constant is the same. Although it is illegal, a single semicolon terminates a character:constant; and for the same reason it terminates a comment, to avoid turning the whole program:declaration inside out if the correct terminator is omitted. The reason for doubling dollar:signs is that a single dollar:sign introduces the codes described in the next two paragraphs.

.4 Any character represented in the body of Figure 2-1, if it is acceptable at all by the system as a character value, may be represented by a three character code beginning with a dollar:sign. The second character is a column code from the figure; i.e., any numeral or one of the letters from A through F. The third character is any character from the body of the figure that can be recognized by the compiler. The character specified by such a code is the one at the intersection of the column designated by the column code and the row in which the third character is found. For example, the percent mark can be represented by any of several three character codes, including these two:

    $25

    $2U

.5 Within a character:constant, there is a recognition mode for letters. Initially, the mode is "general", in which all characters, including uppercase and lowercase letters, and the three-character codes are recognized as described above. The mode can be changed to "lowercase", however, by including the two-character mode code consisting of dollar:sign followed by uppercase or lowercase L. All letters following such a mode code in a character:constant, regardless of the case used, are considered to be in lowercase. The two-character mode consisting of dollar:sign followed by uppercase or lowercase U sets the "uppercase" mode, in which all letters are considered uppercase. The three-character codes pevail, without changing the mode, regardless of the mode. Hence, the appropriate case can be specified for one letter in a stream of letters. For example, here are four character:constants with the value "De Gaulle":

    'De Gaulle'

    'D$6E G$6A$7U$6L$6L$6E'

    'D$LE $4GAULLE'

'Sudslesu gSlaulle' (none of these are ones)

.6 If the count is present in a character:constant, there must be no spaces between the count and the first prime, and the count gives the number of concatenated repetitions of the character values represented within the primes. Examples:

2'TOM' is equivalent to 'TOMTOM'

0'*' is equivalent to '**********'

3' ' is equivalent to '   '

.7 Notice that it is indeed the values that are repeated, not the characters making up the constant before evaluation. Thus, 2'TsLOM' is equivalent to 'TomTom'; it is not equivalent to 'Tomtom'.

.8 The system may impose a limit on the number of characters in strings representable by character:constants, character:variables, or character:formulas. The size of a character:constant is the number of characters represented in the value; not the number of characters between the primes.

(equ171)

```
                                1
                                2
   pattern:constant  ::=        3    B    count    '
pattern:digit    '
                                4
                                5
```

(equ172)

pattern		pattern:digit	order
0 0 0 0 0		0	
0 0 0 0 1		1	1
0 0 0 1 0		2	
0 0 0 1 1		3	
0 0 1 0 0		4	
0 0 1 0 1		5	
0 0 1 1 0		6	
0 0 1 1 1		7	3
0 1 0 0 0		8	
0 1 0 0 1		9	
0 1 0 1 0		A	
0 1 0 1 1		B	
0 1 1 0 0		C	
0 1 1 0 1		D	
0 1 1 1 0	pattern:digit ::=	E	
0 1 1 1 1		F	4
1 0 0 0 0		G	
1 0 0 0 1		H	
1 0 0 1 0		I	
1 0 0 1 1		J	
1 0 1 0 0		K	
1 0 1 0 1		L	
1 0 1 1 0		M	
1 0 1 1 1		N	
1 1 0 0 0		O	
1 1 0 0 1		P	
1 1 0 1 0		Q	
1 1 0 1 1		R	
1 1 1 0 0		S	
1 1 1 0 1		T	
1 1 1 1 0		U	
1 1 1 1 1		V	5

.9 Pattern:constants directly represent values consisting of strings of bits. (Various variables and formulas also represent bit values.) The numeral to the left of the B in the pattern:constant is the "order" of the constant and controls the possible pattern:digits and affects their meanings. These relationships are displayed in the box above wherein pattern:digit is defined. The right column contains the possible orders. The pattern:digits are displayed in the center in braces. The permissible pattern:digits are only those on the line with or above the selected order. For example, if the pattern is of order 4, only F and the 15 pattern:digits above F are permitted as part of this particular pattern:constant. The meaning of each pattern:digit is given in the column on the left, but these are also affected by the order. If the order is n, then the n rightmost bits of each pattern represent the meanings of the corresponding pattern:digits. The optional count gives the number of concatenated repetitions of the pattern:digits enclosed in primes. No spaces are permitted anywhere within this structure.

.10 The meaning of a pattern:constant is the string of bits resulting from the concatenation of the strings of bits (as modified by the order) represented by each pattern:digit. The size of the pattern:constant is the number of bits in the string and may be obtained by multiplying the order times the count (assumed to be 1 if not specified) times the number of characters inside the primes. In the following examples, a pattern:constant on the left is shown with the bit string it represents on the right:

```
4B'7CF03'          01111100111100000011
3B'3120'                   011001010000
1B6'10'                    101010101010
5B2'R'                       1101111011
```

(equ157).Grab9;

```
                              integer:constant
                              fixed:constant
numeric:constant  ::=         floating:constant
                              status:constant
                              qualified:status:constant
```

(equ123)

```
integer:constant  ::=  number
```

(equ81)

```
                                              number   E    +
    scale                                                   -
       floating:constant  ::=      number    .              +
    M    +    scale                                            E
    -    scale                     number    .    number
```

(equ194)

```
       scale  ::=  number
```

(equ77)

```
                               number     .               E    +
    scale    A    +    scale
       fixed:constant  ::=
    -                  -
                               number     .    number
```

(equ222)

```
       status:constant  ::=  V(  status  )
```

(equ221)

```
                       primitive
       status  ::=     name
                       letter
```

(equ187)

```
                                                   status:list:name
                                                   item:name
     qualified:status:constant  ::=  V(       table:name
: status  )
                                                   procedure:name

alternate:entrance:name
```

.11 Numeric:constants represent numeric values. (There are also numeric:variables and numeric:formulas.) Numeric:constants, as well as numeric:variables and numeric:formulas, are described in terms of their three possible modes of representation; as integer values, fixed values, and floating values. The compiler may represent constants in modes other than those indicated by the program:declaration; as long as the overall effect of the program:declaration is not compromised. (This principle applies in general; i.e., the compiler can do things differently as long as the result is the same.) Suppose, for example, an integer:constant is used in a context that requires it to be converted to a floating value. It is far more efficient for that conversion to be done once, at compile time, instead of each time the code executed

.12 An integer value is a numeric value represented as a whole number without a fractional part, but treated as if it had a fractional part with value zero to infinite precision. In this manual, precision means the number of bits to the right of the point in binary representations of numeric values. A number used as an integer:constant represents an unsigned integer value. The size of an integer:constant is the number of bits needed to represent the value; from the leading one bit to the units position, inclusive (value zero has size 1). No spaces are permitted in an integer:constant. The system may impose a limit on sizes of integer values.

.13 Floating values v are represented within the computer by three parts, the significand s, the radix r, and the exrad e, having the following relationships (with regard to the absolute value):

v = s X r

s = 0 or m s < m X r

.14 The radix r and the minimum value m are fixed in any system. Therefore, only the significand and the exrad are saved as representations of a floating value. For a negative value (not a constant), a minus sign is also saved with the significand. Regardless of the system values of r and m, we assume that r = 2 and m is one-half. The language permits inquiry into the values of significands and exrads based on radix and minimum of these values. Therefore, with respect to value, internal representation of floating values exhibits (so far as the programmer can see from results) the relationships:

$$v = s \times 2$$

$$s = 0 \text{ or } 1/2 \quad s \quad 1$$

.15 Floating:constants are written with the assumption that, externally, r = 10, and there is no m. Thus, the value of a floating:constant is given as:

$$v = s \times 10$$

.16 A floating:constant must not contain any spaces. In the syntactic equation for a floating:constant, the number (or numbers) and the decimal:point (if present) give the value of the external significand. The scale (with or without its plus:sign or minus:sign) following E gives an exrad (exponent of the radix) to be used as a power of ten multiplier. If the exrad is zero, it and the E can be omitted. To be a floating:constant, the symbol must contain a decimal:point, or a scale as exrad, or both. It must not contain an A; that would make it a fixed:constant.

.17 A floating:constant can contain information relating to the precision of its internal representation. The scale following M gives the minimum number of magnitude bits in the significand of the internal representation. In most systems, there are one or two or, at most, a very few modes of representation of floating values. If the scale following M is greater than the maximum number of magnitude bits in any of the system-dependent modes of representing floating values, the floating:constant is in error. Otherwise, the compiler chooses the mode with the smallest number of magnitude bits in the significand at least as large as the scale following M. If there is a choice of exrad size also, the compiler chooses one that can encompass the value of the floating:constant. These sizes are based on the numbers of bits in the actual representations, not on what may be a fictional assumption that the radix is 2. If the M and its following scale are omitted, the compiler chooses its normal mode of floating representation or one that can contain the value.

.18 A fixed value is an approximate numeric value. Within the computer, it is represented as a string of bits with an assummed binary point within or to the left or right of the string. The number of bits in the string, not counting a sign bit if there is one, is the size of the fixed value. The number of bits after the point (positive or negative, larger or smaller than the size) is the precision of the fixed value.

.19 A fixed:constant is seen, in the syntactic equation above, to be an integer:constant or a floating:constant (without an M and its scale) followed by the letter A and a scale. The A and its scale are essential to make the form a fixed:constant. Spaces are not allowed anywhere within a fixed:constant. All that precedes the A determines the value of the fixed:constant. All that precedes the A determines the value of the fixed:constant (which may then be truncated on the right). The scale after the A tells how many bits there are after the point. (If the scale is negative, the bits don't even come as far to the right as the point). The size of the constant is the number of bits from the leftmost one-bit to the number after the point as specified by the scale after A, inclusive. Here are some fixed:constants, their values, their sizes, and their precisions:

(tab2)

fixed:constant	value	size	precision
19A0	19	5	0
19A3	19	8	3
19A-2	16	3	-2
2.3A0	2	2	0
2.3A-1	2	1	-1
2.3A2	2.25	4	2
2.3a5	2.28125	7	5
2.3A6	2.96875	8	6

.20 There must be no spaces within a fixed:constant. The system may impose a size limitation on fixed values.

.21 Integer:constants, floating:constants, and fixed:constants cannot have embedded spaces and cannot have negative values. Both of these characteristics are changed for status:constants and qualified:status:constants. In status:constants and qualified:status:constants, there must be no spaces within the status, within the qualifying name, or between the V and the left:parenthesis. There may be spaces elsewhere within such constants.

.22 Status:constants and qualified:status:constants represent constant integer values. How they become associated with these values and how they may be used are explained elsewhere. In distinction to integer:constants, which can only stand for zero and positive integer values, status:constants and qualified:status:constants can also stand for unvarying negative integer values.

.DefSyn[MonoSpace]=M; .DefSyn[Slant]=S; .DefSyn[BoldFace]=B;

2.9 Computer Representation of Constants and Variables

JOVIAL is designed to be compatible with binary computers, machines in which numeric and other values are represented as strings of binary digits, ones and zeros. The bits (binary digits) of a computer are organized in a hierarchical structure. A compiler may impose a different structure on the computer, but for reasons of efficiency it usually adopts a structure identical to or at least compatible with the structure of the machine. The structure discussed in this section is the system structure; i.e., the structure presented to the programmer by the combination of a particular computer and a particular JOVIAL compiler that produces object code for that computer.

.1 JOVIAL program:declarations are not completely independent of the system. The extent of dependence, however, is related to the use of certain language features. Dependence is increased by the use of features, such as pattern:constants and BIT, that relate to bit representation or those, such as LOC, that relate to system structure. The value of a pattern:constant is completely independent of the system, but its use implies knowledge of the representation of other data. It is that knowledge, built into te program:declaration, that is system dependent.

.2 Even if such deliberate system dependence is avoided, the programmer must still have knowledge of structure and representation in his system so that he may know the limitations on precision, how his tables must be structured, and how to avoid gross inefficiencies. For example, in processing long strings of character data, it is often much faster to examine and manipulate them in word-size, instead of byte-size, hunks.

.3 A "byte" is a group of bits often used to represent one character of data. The number of bits in a byte is system dependent. Although JOVIAL permits some leeway in positioning bytes, there are usually preferred positions. When referring to these preferred positions, we often use the term "byte boundary".

.4 A "word" is a system-dependent grouping of bits convenient for describing data allocation. Entries and tables are allocated in terms of words. Data are overlaid in terms of words. The maximum sizes of numeric values may, but need not, be related to words. Word boundaries usually correspond to some of the byte boundaries.

.5 The "basic addressable unit" is the group of bits corresponding to each machine location. In many machines, the basic addressable unit is the word. In others, it is the byte. If it is the word, each value of the location counter refers to a unique word. If the basic addressable unit is the byte, each location value refers to a unique byte. In these latter circumstances, it often happens that adresses are somewhat restricted. For instance, it may be permitted to refer to a string of characters starting in any byte, or to double-precision floating values starting only in bytes with locations divisible by 8.

.6 Integer and fixed values are represented in binary as strings of bits. The number of bits used to represent the magnitude of a value is known as its size and is (in most cases) under the control of the programmer. The position of the binary point is understood and takes up no space. For signed values, the sign bit is an additional bit not counted in the size of the value. For purposes of the use of BIT, the sign bit is considered to lie just to be left of the most significant bit accounted for by the size of the value. The maximum permissible size of an integer or fixed value is system dependent. The maximum size of a signed integer or fixed value is one less than this system-dependent size and the places where unsigned values of maximum size may be used are restricted; i.e., they must not be used in conjunction with any arithmetic:operators, nor with the four nnnrxlldtrhb rdl'thnn'l:npdr'tnrr <, >, <=, >=, and when used with the symmetric relational:operators (= and <>) the other operand must not be signed.

.7 The compiler determines the sizes of constants. The programmer usually supplies the sizes of variables. The size does not include the sign bit for signed data. For unpacked or medium packed data, there may be more bits in the space allocated for an item than are specified by the programmer. Whether or how these extra bits are used is system dependent, but in any case they are known as "filler bits". The sign bit, if there is one, and any filler bits are to the left of the magnitude bits. It depends on the system whether the sign bit is to the left or right of the filler bits.

.8 The meanings of bit values 0 and 1 are not stipulated, but in most implementations 0 stands for 0 and 1 for 1 in positive values. For negative values, there is considerable variation. All the following are known and acceptable representations of -12 in an unpacked, signed, integer item declared to be four bits long:

111111111111111111111111111111111111111111110011

100000000000000000000000000000001100

10100

.9 Floating values are represented by two numbers, both signed. The significand contains the significant digits of the value and the exrad is the exponent of the understood radix. Each system has a standard mode of representing floating values, known as "single precision", with a specified number of bits in the significand and a specified number in the exrad. Many systems have one or a few additional modes in which there are more bits in the significand, the exrad, or both. If there is more than one mode, the programmer can usually choose the mode for each floating value. In the absence of an indication of such choice, the compiler will usually choose single precision. The radix is an implicit constant having a system-dependent value.

.10 Character values are represented by strings of bytes, each byte consisting of a string of bits. The number of bits in a byte is system dependent. The number of bytes used to represent a character value is under control of the programmer, but there is a system-dependent maximum.

.11 A character item that fits in one word is always stored in one word, by the compiler. By use of a specified:table:declaration, the programmer may override this rule. If it is not densely packed, a character item always starts at a byte boundary. If it crosses a word boundary, a character item always starts at a byte boundary. The programmer must not attempt to override this rule.

.12 An entry variable whose relevent table:declaration does not describe it as being of some other type is a bit variable. It is merely the string of bits, of a size corresponding to the number of words in an entry, representing the entry.

2.10 Spaces, Comments

The syntactic structures of all symbols have now been explained, as well as the places where spaces are permitted or prohibited within them. All further structures that go to make up a program:declaration are composed of strings of symbols. It is always permitted to place one or more spaces between symbols. It is sometimes required to put at least one space between symbols. The criterion is to avoid ambiguity. Comments can often replace required spaces.

.1 Spaces are required in many situations to enable the compiler to detect the end of one symbol and the beginning of the next. Generally, at least one space is required between two symbols of any class except ideograms, but including the quotation:mark. The rule is exhibited in detail in the following table. The rows are labelled with the ending signs of the left symbol of a pair of symbols. The columns are labelled with the beginning signs of the right symbol of a pair. "SR" at the intersection of row and column indicates that at least one space is required between the pair of symbols:

(tab3)

Left symbol ends in	Right symbol starts with: numeral	letter	$	'	.	"
numeral	SR	SR	SR	SR	SR	
letter	SR	SR	SR	SR		
$	SR	SR	SR	SR		
'	SR	SR	SR	SR		
.	SR	SR				
"						SR

.2 A comment may occur between symbols. However, it must not occur within a definition nor within any constant, such as a status:constant or a character:constant. A comment may be used instead of the required space between symbols unless use of the comment would cause the occurrence of two quotation:marks in succession. In fact, only the use of a comment can bring about the situation indicated by the lower right corner of the table above. Introduction of a comment between symbols where a space is permitted but not required may then require a space to prevent the comment from interfering with another symbol.

.3 A comment must not be used where the next structure required or permitted by the syntax is a definition. That is, a comment must not follow the define:name or a right:parenthesis in a define:declaration. And a comment must not follow a left:parenthesis or a comma in a definition:invocation. A comment, as defined above, must not occur in a definition delimited by quotation:marks.

,DefSyn[MonoSpace]=M; ,DefSyn[Slant]=S; ,DefSyn[BoldFace]=B;

Chapter 2

ELEMENTS

2.1 Introduction

A program:declaration written in JOVIAL consists, basically, of statements and declarations. The statements specify the computations to be performed with arbitrarily named data. Simple:statements can be grouped together into compound:statements in order to help in specifying the order of computations. Among the declarations are data:declarations and processing:declarations. The data:declarations name and describe the data on which the program is to operate, including inputs, intermediate results, and final results. The processing:declarations generally contain statements and other declarations. They specify computations, but they differ from statements in that the computations must be performed only when the particular processing:declaration is specifically invoked by name. In addition to statements and declarations, there are directives which serve various purposes. They designate externally defined names the compiler is expected to recognize, they control selective compilation of various statements and declarations, and they provide information the compiler needs in order to optimize the object code. The statements, declarations, and directives are composed of symbols, which are the words of the JOVIAL language. These symbols are, in turn, composed of the signs that constitute the JOVIAL alphabet.

.1 The general order in which the elements of a program:declaration are introduced in the preceding paragraph represents the general order in which one looks up definitions when trying to clear up a question. The definitions in this manual are introduced, however, in the opposite order. Such arrangements lead to complaints that one must "read the book backwards." This comment arises from the process of looking up a form in the table of contents, turning then to the late chapter where it is defined in terms of earlier defined forms. These, more elementary, forms are then found, via the table of contents, in an earlier chapter. And so forth. Nevertheless, the document is arranged for the use of a reader rather than for reference. Difficult as this may be for reference use, the opposite arrangement is much more difficult for a reader.

.2 An index-glossary is included which facilitates reference. The index-glossary answers many questions directly. In other cases, it references syntax equations and sections by number.

2.2 Spaces and Spaces

It is important to distinguish between a space, an element of JOVIAL, and a space, an element of our descriptive language. JOVIAL is written using symbols, the words of the language. The symbols are composed of signs, the elements of the JOVIAL alphabet. In general, symbols do not contain spaces. The exceptions are pointed out in Section 2.5.2, with respect to comment, and in Section 2.8.2, with respect to character:constraints. In general, symbols are separated by spaces. Again the exceptions are noted in Section 2.10; however, these exceptions are permissive; i.e., it is always correct to put spaces between symbols.

.1 The following example is wrong:

```
PLXMPY ( 1. 375, -. 75, 5 ., 7.3 : REAL,
IMAG ) ;
```

.2 The following examples are right:

```
a. BEGIN 1, 3, +5, - 7 END

b. SL:PLXMPY(1.375,-.75,5.,7.3:REAL,IMAG);

c. SL : PLXMPY ( 1.375 , - .75 , 5. , 7.3 :
REAL , IMAG ) ;
```

.3 In defining and explaining signs and symbols, any spaces included in the metalanguage formulas are not meant to be included in the definition. The phrase "string of" implies that there are to be no spaces between the elements strung together. Similarly, phrases such as "followed by", "enclosed in", and "separated by", imply that there are to be no spaces between the elements concerned. This is the situation (except where explicitly stated to be different) in this chapter, Chapter 2. In Chapter 3 and beyond, the opposite view is maintained with respect to these phrases.

2.3 Signs, Elements of the JOVIAL Alphabet

(equ197)

```
             letter
sign  ::=    numeral
             mark
```

(equ134)

```
                          A
                          B
                          C
                          D
                          E
                          F
                          G
                          H
                          I
                          J
                          K
                          L
           letter  ::=    M
                          N
                          O
                          P
                          Q
                          R
                          S
                          T
                          U
                          V
                          W
                          X
                          Y
                          Z
```

(equ156)

```
                          0
                          1
                          2
                          3
           numeral  ::=   4
                          5
                          6
                          7
                          8
                          9
```

(equ144)

	+	plus:sign
	-	minus:sign
	*	astrerisk
	/	slash
	\	back:slash
	&	ampersand
	>	greater:than:sign
	<	less:than:sign
	=	equals:sign
	@	at:sign
mark ::=	.	decimal:point
	:	colon
	,	comma
	;	semicolon
		space
	(	left:parenthesis, parenthesis
	)	right:parenthesis, parenthesis
	[	left:backet, bracket
	]	right:bracket, bracket
	'	prime
	"	quotation:mark
	$	dollar:sign
	!	exclamation:point

.1 Sign means a letter, a numeral or a mark. Letter means one of the 26 letters of the English alphabet, written in the form of a roman capital. Numeral means one of the ten arabic numerals: 0,1,2,3,4,5,6,7,8 or 9. (The slash through the zero is only for the purpose of distinguishing it from the letter O in definitions and examples of JOVIAL.) Sign, letter, and numeral are defined more formally by means of the syntax equations in the boxes at the head of this section. Mark is most easily defined by the formal means of the syntax equation in the box above. The box above also contains a metalinguistic term associated with each mark; this serves to define these terms.

2.4 Symbols, The Words of JOVIAL

(equ233)

```
                      primitive
                      ideogram
                      name
                      letter:control:variable
    symbol  ::=       abbreviation
                      number
                      constant
                      comment
                      directive:key
                      status
```

.1 The symbols or words of the JOVIAL language are composed of strings of signs, in some cases a single sign. Most symbols do not contain spaces. In fact, spaces serve to separate symbols from one another.

2.5 PRIMITIVE, Ideogram, Directive:Key, Comment

(equ178)

```
                                ABS
                                ALL
                                ALT          NENT
                                AND          NOT
                                BEGIN        NULL
                                FIT          NWDSEN
                                BLOCK        OR
                                BY           OVERLAY
                                BYTE         PROC
                                DEF          PROGRAM
                                DEFINE       REF
                                DIRECT       REMQUO
                                DSIZE        RESERVE
                                ELSE         RETURN
                                END          SHIFT
              primitive  ::=    ENTER        SIG
                                EQV          SIGNED
                                EXIT         SIGNUM
                                FOR          SIZE
                                FORM         STATUS
                                FORMAT       STOP
                                FRAC         SWITCH
                                GOTO         TABLE
                                IF           TEST
                                IN           THEN
                                INT          TYPE
                                ISIZE        UNTIL
                                ITEM         WHILE
                                JOVIAL       XOR
                                LOC          XRAD
                                NAME         ZAP
```

(equ106)

```
                     +
                     -
                     /
                     **
                     \
                     &
                     =
                     ==
                     <
                     >
                     <=
                     >=
ideogram  ::=        <>
                     .
                     ,
                     :
                     ;
                     !
                     "
                     '
                     (
                     )
                     [
                     ]
                     @
                     @@
```

(equ64)

```
                          !COMPOOL
                          !SKIP
                          !BEGIN
                          !END
                          !TRACE
                          !COPY
                          !ABNORMAL
                          !SETS
directive:key  ::=        !USES
                          !POINTER
                          !ORDER
                          !RECURSIVE
                          !TIME
                          !SPACE
                          !LINKAGE
                          !INTERFERENCE
                          !FREQUENCY
```

(equ32)

```
comment  ::=  "  character  "
```

(equ25)

```
                      sign
character  ::=
                      system:dependent:character
```

.1 Primitives may be considered the key words of the JOVIAL language. They are generally used to give the primary meaning of a statement or declaration, although some are used for second purposes. Ideograms are generally used as arithmetic:operators, as relational:operators, and for purposes such as grouping, separating, and terminating. Directive:keys are used to state the primary meanings of directives. Comments can be used to annotate a program:declaration: explaining to readers (and often the original programmer) what is going on.

.2 Notice that a comment is delimited by quotation:marks. Therefore, spaces are permitted within a comment, but a quotation:mark is not permitted within a comment. Also, a semicolon is not permitted within a comment. The reason for this is to permit some recovery in case a delimiting quotation:mark is left off a comment. If the comment were not then terminated by the next semicolon, the entire remainder of the program:declaration would be turned inside out; the comments being interchanged with the statements and declarations. Even with this rule, failure to terminate a comment can lead to disaster. If an END is swallowed up, the entire program structure can be disarrayed.

.3 The system:dependent:characters that can be included in comments (and other structures) are simply those characters, other than JOVIAL signs, that the particular system and compiler can read and write.

.4 Notice that primitives, ideograms, and directive:keys do not contain spaces. Spaces are significant in a program:declaration; usually in that they separate symbols. Comments, on the other hand, may contain spaces. This permits easier reading and writing of the commentary. The quotation:marks delimiting the comment provide the necessary grouping so that the spaces do not cause trouble.

.DefSyn[MonoSpace]=M; .DefSyn[Slant]=S; .DefSyn[BoldFace]=B;

2.6 Abbreviation, Letter:Control:Variable, Name

(equ1)

```
abbreviation  :=  letter
```

(equ135)

```
letter:control:variable  ::=  letter
```

(equ145)

```
                                letter
name  ::=     letter            numeral
                $                 $
                                  '
```

.1 Abbreviations are specific letters having specific meanings in specific contexts, usually data:declarations. The specific uses are documented later on without, usually, calling the letter an abbreviation.

.2 The letter:control:variable is a special variable having meaning only within a loop:statement and passing out of existence when the loop:statement is not being executed. It is explained more fully in connection with explanation of the loop:statement.

.3 Regardless of the syntax in the box above, a name must not be the same as any primitive. Notice that a name must include at least two signs. The use of the dollar:sign is system dependent. That is, it provides a means whereby a name can be designated to have some special meaning in relation to the system in which the compiler is embedded. Such special meanings are outside the scope of this manual, however, and names containing dollar:signs are considered the same as other names herein. Names do not contain spaces. An embedded space would change a name into two names or other symbols.

2.7 Number, Constant, Status

(equ154)

```
number ::= numeral
```

(equ39)

```
                             numeric:constant
constant:formula  ::=        pattern:constant
                             character:constant
```

(equ26)

```
character:constant  ::=  count  '  character  '
```

(equ47)

```
count  ::=  number
```

(equ157)

```
                             integer:constant
                             fixed:constant
numeric:constant  ::=        floating:constant
                             status:constant
                             qualified:status:constant
```

(equ222)

```
status:constant  ::=  V(  status  )
```

(equ187)

```
                                          status:list:name
                                          item:name
      qualified:status:constant  ::=  V(    table:name
   :  status  )
                                          procedure:name

   alternate:entrance:name
```

(equ221)

```
      status  ::=    primitive
                     name
                     letter
```

.1 The above definitions are obviously not complete, in that several kinds of constants mentioned in the box are not yet defined. This discussion is mainly concerned with the use of spaces together with numbers, constants, and statuses as symbols.

.2 A number is a string of numerals, without spaces. In some places, a number can stand alone as a constant. In other places, particularly data:declarations, it stands alone as a symbol but is not considered a constant. In yet other places, a number is part of another symbol. A case in point is the character:constant, defined above. The optional count in a character:constant is a number. (In several places, numbers or other constructs are given new names reminiscent of their uses in those places.)

.3 A character:constant is a symbol. If it begins with a count, there must be no spaces between the count and the first prime. Between the primes, the string of characters may include spaces, but these spaces are significant. They represent part of the value represented by the character:constant. (There are restrictions on the characters permitted in a character:constant, discussed in Section 2.8.2). In a status:constant and a qualified:status:constant, the left:parenthesis, the name, the colon, the status, and the right:parenthesis are all symbols. Spaces are permitted between these elements, but not within the name or the status. Space is not pemitted between V and the left:parenthesis. All other constants are symbols, not containing spaces.

2.8 Constants and Values

(equ39)

```
constant:formula  ::=   numeric:constant
                        pattern:constant
                        character:constant
```

(equ26)

```
character:constant  ::=  count  '  character  '
```

(equ47)

```
count  ::=  number
```

.1 Character:constants are the direct means of representing character values to be manipulated by a program. (Character:variables and character:formulas are indirect means.) The characters acceptable as character values are whatever the system will accept from among those given in the body of Figure 2-1. At least the 59 JOVIAL signs must be accepted. Comparison of Figure 2-1 with Section 2 of USAS X3.4-1968, "USA Standard Code for Information Interchange", shows the graphic characters in identical positions in the two tables. Figure 2-1 includes eight additional columns presently under consideration by standardization bodies. The positions of the characters in the table are the only correspondence. This manual does not require that internal representation be in accordance with USAS X3.4-1968. If, however, JOVIAL program:declarations generate messages for transmission to other systems or process messages received from other systems, these messages are required by other directives to conform to USAS X3.4-1968 in their external representation.

(tab1)

Column	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Column Code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Row																
0			space	0	@	P		p								
1			!	1	A	Q	a	q								
2			"	2	B	R	b	r								
3			#	3	C	S	c	s								
4			$	4	D	T	d	t								
5			%	5	E	U	e	u								
6			&	6	F	V	f	v								
7			'	7	G	W	g	w								
8			(	8	H	X	h	x								
9			)	9	I	Y	i	y								
10			*	:	J	Z	j	z								
11			+	;	K	[	k									
12			,	<	L	\	l									
13			-	=	M	]	m									
14			.	>	N		n									
15			/	?	O		o									

Notes: row 0, column 3: zero
row 1, column 3: one
row 7, column 2: prime, often rendered as a vertical mark in JOVIAL
row 12, column 6: a lowercase letter
row 15, column 4: an uppercase letter

Figure 2-1, Characters

.2 All of the character values indicated in the body of Figure 2-1 can be represented in character:constants (except for system-dependent limitations), Artifices are required, however, to represent some of the values. Any spaces within the delimiting primes, except within a three-character code, represent characters of value "space", Primes, semicolons, and dollar:signs have special meanings, Therefore, in order to represent a single occurrence of one of these signs, two of them are used in succession. If a succession of these signs is desired as part of the value represented by a character:constant, the entire string is doubled. In summary:

2n primes are used to represent n primes,

2n semicolons are used to represent n semicolons,

2n dollar:signs are used to represent n dollar:signs,

.3 The reason for doubling the primes inside a character:constant is that single prime terminates the constant. The reason for doubling semicolons inside a character:constant is the same. Although it is illegal, a single semicolon terminates a character:constant; and for the same reason it terminates a comment, to avoid turning the whole program:declaration inside out if the correct terminator is omitted. The reason for doubling dollar:signs is that a single dollar:sign introduces the codes described in the next two paragraphs.

.4 Any character represented in the body of Figure 2-1, if it is acceptable at all by the system as a character value, may be represented by a three character code beginning with a dollar:sign. The second character is a column code from the figure; i.e., any numeral or one of the letters from A through F. The third character is any character from the body of the figure that can be recognized by the compiler. The character specified by such a code is the one at the intersection of the column designated by the column code and the row in which the third character is found. For example, the percent mark can be represented by any of several three character codes, including these two:

```
$25

$2U
```

.5 Within a character:constant, there is a recognition mode for letters. Initially, the mode is "general", in which all characters, including uppercase and lowercase letters, and the three-character codes are recognized as described above. The mode can be changed to "lowercase", however, by including the two-character mode code consisting of dollar:sign followed by uppercase or lowercase L. All letters following such a mode code in a character:constant, regardless of the case used, are considered to be in lowercase. The two-character mode consisting of dollar:sign followed by uppercase or lowercase U sets the "uppercase" mode, in which all letters are considered uppercase. The three-character codes pevail, without changing the mode, regardless of the mode. Hence, the appropriate case can be specified for one letter in a stream of letters. For example, here are four character:constants with the value "De Gaulle":

```
'De Gaulle'

'D$6E G$6A$7U$6L$6L$6E'

'D$LE $4GAULLE'
```

'sudslesu oSlaulle' (none of these are ones)

.6 If the count is present in a character:constant, there must be no spaces between the count and the first prime, and the count gives the number of concatenated repetitions of the character values represented within the primes. Examples:

2'TOM' is equivalent to 'TOMTOM'

0'*' is equivalent to '**********'

3' ' is equivalent to '   '

.7 Notice that it is indeed the values that are repeated, not the characters making up the constant before evaluation. Thus, 2'T$LOM' is equivalent to 'TomTom'; it is not equivalent to 'Tomtom'.

.8 The system may impose a limit on the number of characters in strings representable by character:constants, character:variables, or character:formulas. The size of a character:constant is the number of characters represented in the value; not the number of characters between the primes.

(equ171)

```
                                 1
                                 2
    pattern:constant  ::=        3    B    count    '
pattern:digit    '
                                 4
                                 5
```

(equi72)

pattern		pattern:digit	order
0 0 0 0 0		0	
0 0 0 0 1		1	1
0 0 0 1 0		2	
0 0 0 1 1		3	
0 0 1 0 0		4	
0 0 1 0 1		5	
0 0 1 1 0		6	
0 0 1 1 1		7	3
0 1 0 0 0		8	
0 1 0 0 1		9	
0 1 0 1 0		A	
0 1 0 1 1		B	
0 1 1 0 0		C	
0 1 1 0 1		D	
0 1 1 1 0	pattern:digit ::=	E	
0 1 1 1 1		F	4
1 0 0 0 0		G	
1 0 0 0 1		H	
1 0 0 1 0		I	
1 0 0 1 1		J	
1 0 1 0 0		K	
1 0 1 0 1		L	
1 0 1 1 0		M	
1 0 1 1 1		N	
1 1 0 0 0		O	
1 1 0 0 1		P	
1 1 0 1 0		Q	
1 1 0 1 1		R	
1 1 1 0 0		S	
1 1 1 0 1		T	
1 1 1 1 0		U	
1 1 1 1 1		V	5

.9 Pattern:constants directly represent values consisting of strings of bits. (Various variables and formulas also represent bit values.) The numeral to the left of the B in the pattern:constant is the "order" of the constant and controls the possible pattern:digits and affects their meanings. These relationships are displayed in the box above wherein pattern:digit is defined. The right column contains the possible orders. The pattern:digits are displayed in the center in braces. The permissible pattern:digits are only those on the line with or above the selected order. For example, if the pattern is of order 4, only F and the 15 pattern:digits above F are permitted as part of this particular pattern:constant. The meaning of each pattern:digit is given in the column on the left, but these are also affected by the order. If the order is n, then the n rightmost bits of each pattern represent the meanings of the corresponding pattern:digits. The optional count gives the number of concatenated repetitions of the pattern:digits enclosed in primes. No spaces are permitted anywhere within this structure.

.10 The meaning of a pattern:constant is the string of bits resulting from the concatenation of the strings of bits (as modified by the order) represented by each pattern:digit. The size of the pattern:constant is the number of bits in the string and may be obtained by multiplying the order times the count (assumed to be 1 if not specified) times the number of characters inside the primes. In the following examples, a pattern:constant on the left is shown with the bit string it represents on the right:

4B'7CF03'	01111100111100000011
3B'3120'	011001010000
1B6'10'	101010101010
5B2'R'	1101111011

(equ157)

```
numeric:constant  ::=  integer:constant
                       fixed:constant
                       floating:constant
                       status:constant
                       qualified:status:constant
```

(equ123)

```
integer:constant  ::=  number
```

(equ81)

```
                                              number   E    +
     scale                                                     -
         floating:constant  ::=       number   ,          +
     M    +   scale                                              E
     -    scale
                                      number   ,   number
```

(equ194)

```
         scale  ::=  number
```

(equ77)

```
                                 number    ,              E    +
     scale    A    +   scale
         fixed:constant  ::=
     -                 -
                                 number    ,    number
```

(equ222)

```
         status:constant  ::=  V(  status  )
```

(equ221)

```
                          primitive
         status  ::=      name
                          letter
```

(equ187)

```
                                                    status:list:name
                                                    item:name
    qualified:status:constant  ::=  V(    table:name
: status  )
                                                    procedure:name

alternate:entrance:name
```

,11 Numeric:constants represent numeric values. (There are also numeric:variables and numeric:formulas.) Numeric:constants, as well as numeric:variables and numeric:formulas, are described in terms of their three possible modes of representation; as integer values, fixed values, and floating values. The compiler may represent constants in modes other than those indicated by the program:declaration; as long as the overall effect of the program:declaration is not compromised. (This principle applies in general; i.e., the compiler can do things differently as long as the result is the same.) Suppose, for example, an integer:constant is used in a context that requires it to be converted to a floating value. It is far more efficient for that conversion to be done once, at compile time, instead of each time the code executed

,12 An integer value is a numeric value represented as a whole number without a fractional part, but treated as if it had a fractional part with value zero to infinite precision. In this manual, precision means the number of bits to the right of the point in binary representations of numeric values. A number used as an integer:constant represents an unsigned integer value. The size of an integer:constant is the number of bits needed to represent the value; from the leading one bit to the units position, inclusive (value zero has size 1). No spaces are permitted in an integer:constant. The system may impose a limit on sizes of integer values.

,13 Floating values v are represented within the computer by three parts, the significand s, the radix r, and the exrad e, having the following relationships (with regard to the absolute value):

v = s X r

s = 0 or m s < m x r

,14 The radix r and the minimum value m are fixed in any system. Therefore, only the significand and the exrad are saved as representations of a floating value. For a negative value (not a constant), a minus sign is also saved with the significand. Regardless of the system values of r and m, we assume that r = 2 and m is one-half. The language permits inquiry into the values of significands and exrads based on radix and minimum of these values. Therefore, with respect to value, internal representation of floating values exhibits (so far as the programmer can see from results) the relationships:

v = s x 2

s = 0 or 1/2   s   1

.15 Floating:constants are written with the assumption that, externally, r = 10, and there is no m. Thus, the value of a floating:constant is given as:

v = s x 10

.16 A floating:constant must not contain any spaces. In the syntactic equation for a floating:constant, the number (or numbers) and the decimal:point (if present) give the value of the external significand. The scale (with or without its plus:sign or minus:sign) following E gives an exrad (exponent of the radix) to be used as a power of ten multiplier. If the exrad is zero, it and the E can be omitted. To be a floating:constant, the symbol must contain a decimal:point, or a scale as exrad, or both. It must not contain an A; that would make it a fixed:constant.

.17 A floating:constant can contain information relating to the precision of its internal representation. The scale following M gives the minimum number of magnitude bits in the significand of the internal representation. In most systems, there are one or two or, at most, a very few modes of representation of floating values. If the scale following M is greater than the maximum number of magnitude bits in any of the system-dependent modes of representing floating values, the floating:constant is in error. Otherwise, the compiler chooses the mode with the smallest number of magnitude bits in the significand at least as large as the scale following M. If there is a choice of exrad size also, the compiler chooses one that can encompass the value of the floating:constant. These sizes are based on the numbers of bits in the actual representations, not on what may be a fictional assumption that the radix is 2. If the M and its following scale are omitted, the compiler chooses its normal mode of floating representation or one that can contain the value.

.18 A fixed value is an approximate numeric value. Within the computer, it is represented as a string of bits with an assummed binary point within or to the left or right of the string. The number of bits in the string, not counting a sign bit if there is one, is the size of the fixed value. The number of bits after the point (positive or negative, larger or smaller than the size) is the precision of the fixed value.

.19 A fixed:constant is seen, in the syntactic equation above, to be an integer:constant or a floating:constant (without an M and its scale) followed by the letter A and a scale. The A and its scale are essential to make the form a fixed:constant. Spaces are not allowed anywhere within a fixed:constant. All that precedes the A determines the value of the fixed:constant. All that precedes the A determines the value of the fixed:constant (which may then be truncated on the right). The scale after the A tells how many bits there are after the point. (If the scale is negative, the bits don't even come as far to the right as the point). The size of the constant is the number of bits from the leftmost one-bit to the number after the point as specified by the scale after A, inclusive. Here are some fixed:constants, their values, their sizes, and their precisions:

(tab2)

fixed:constant	value	size	precision
19A0	19	5	0
19A3	19	8	3
19A-2	16	3	-2
2.3A0	2	2	0
2.3A-1	2	1	-1
2.3A2	2.25	4	2
2.3a5	2.28125	7	5
2.3A6	2.96875	8	6

.20 There must be no spaces within a fixed:constant. The system may impose a size limitation on fixed values.

.21 Integer:constants, floating:constants, and fixed:constants cannot have embedded spaces and cannot have negative values. Both of these characteristics are changed for status:constants and qualified:status:constants. In status:constants and qualified:status:constants, there must be no spaces within the status, within the qualifying name, or between the V and the left:parenthesis. There may be spaces elsewhere within such constants.

.22 Rt'ttr:bnnrt'nts and qualified:status:constants represent constant integer values. How they become associated with these values and how they may be used are explained elsewhere. In distinction to integer:constants, which can only stand for zero and positive integer values, status:constants and qualified:status:constants can also stand for unvarying negative integer values.

.DefSyn[MonoSpace]=M; .DefSyn[Slant]=S; .DefSyn[BoldFace]=B;

2,9 Computer Representation of Constants and Variables

JOVIAL is designed to be compatible with binary computers, machines in which numeric and other values are represented as strings of binary digits, ones and zeros, The bits (binary digits) of a computer are organized in a hierarchical structure, A compiler may impose a different structure on the computer, but for reasons of efficiency it usually adopts a structure identical to or at least compatible with the structure of the machine, The structure discussed in this section is the system structure; i,e,, the structure presented to the programmer by the combination of a particular computer and a particular JOVIAL compiler that produces object code for that computer,

,1 JOVIAL program:declarations are not completely independent of the system, The extent of dependence, however, is related to the use of certain language features, Dependence is increased by the use of features, such as pattern:constants and BIT, that relate to bit representation or those, such as LOC, that relate to system structure, The value of a pattern:constant is completely independent of the system, but its use implies knowledge of the representation of other data, It is that knowledge, built into te program:declaration, that is system dependent,

,2 Even if such deliberate system dependence is avoided, the programmer must still have knowledge of structure and representation in his system so that he may know the limitations on precision, how his tables must be structured, and how to avoid gross inefficiencies, For example, in processing long strings of character data, it is often much faster to examine and manipulate them in word-size, instead of byte-size, hunks,

,3 A "byte" is a group of bits often used to represent one character of data, The number of bits in a byte is system dependent, Although JOVIAL permits some leeway in positioning bytes, there are usually preferred positions, When referring to these preferred positions, we often use the term "byte boundary",

,4 A "word" is a system-dependent grouping of bits convenient for describing data allocation, Entries and tables are allocated in terms of words, Data are overlaid in terms of words, The maximum sizes of numeric values may, but need not, be related to words, Word boundaries usually correspond to some of the byte boundaries,

,5 The "basic addressable unit" is the group of bits corresponding to each machine location, In many machines, the basic addressable unit is the word, In others, it is the byte, If it is the word, each value of the location counter refers to a unique word. If the basic addressable unit is the byte, each location value refers to a unique byte. In these latter circumstances, it often happens that adresses are somewhat restricted. For instance, it may be permitted to refer to a string of characters starting in any byte, or to double-precision floating values starting only in bytes with locations divisible by 8.

.6 Integer and fixed values are represented in binary as strings of bits. The number of bits used to represent the magnitude of a value is known as its size and is (in most cases) under the control of the programmer. The position of the binary point is understood and takes up no space. For signed values, the sign bit is an additional bit not counted in the size of the value. For purposes of the use of BIT, the sign bit is considered to lie just to be left of the most significant bit accounted for by the size of the value. The maximum permissible size of an integer or fixed value is system dependent. The maximum size of a signed integer or fixed value is one less than this system-dependent size and the places where unsigned values of maximum size may be used are restricted; i.e., they must not be used in conjunction with any arithmetic:operators, nor with the four nonsymmetric relational:operators <, >, <=, >=, and when used with the symmetric relational:operators (= and <>) the other operand must not be signed.

.7 The compiler determines the sizes of constants. The programmer usually supplies the sizes of variables. The size does not include the sign bit for signed data. For unpacked or medium packed data, there may be more bits in the space allocated for an item than are specified by the programmer. Whether or how these extra bits are used is system dependent, but in any case they are known as "filler bits". The sign bit, if there is one, and any filler bits are to the left of the magnitude bits. It depends on the system whether the sign bit is to the left or right of the filler bits.

.8 The meanings of bit values 0 and 1 are not stipulated, but in most implementations 0 stands for 0 and 1 for 1 in positive values. For negative values, there is considerable variation. All the following are known and acceptable representations of -12 in an unpacked, signed, integer item declared to be four bits long:

```
111111111111111111111111111111111111111110011

                  100000000000000000000000000001100

                                              10100
```

.9 Floating values are represented by two numbers, both signed. The significand contains the significant digits of the value and the exrad is the exponent of the understood radix. Each system has a standard mode of representing floating values, known as "single precision", with a specified number of bits in the significand and a specified number in the exrad. Many systems have one or a few additional modes in which there are more bits in the significand, the exrad, or both. If there is more than one mode, the programmer can usually choose the mode for each floating value. In the absence of an indication of such choice, the compiler will usually choose single precision. The radix is an implicit constant having a system-dependent value.

.10 Character values are represented by strings of bytes, each byte consisting of a string of bits. The number of bits in a byte is system dependent. The number of bytes used to represent a character value is under control of the programmer, but there is a system-dependent maximum.

.11 A character item that fits in one word is always stored in one word, by the compiler. By use of a specified:table:declaration, the programmer may override this rule. If it is not densely packed, a character item always starts at a byte boundary. If it crosses a word boundary, a character item always starts at a byte boundary. The programmer must not attempt to override this rule.

.12 An entry variable whose relevent table:declaration does not describe it as being of some other type is a bit variable. It is merely the string of bits, of a size corresponding to the number of words in an entry, representing the entry.

2.10 Spaces, Comments

The syntactic structures of all symbols have now been explained, as well as the places where spaces are permitted or prohibited within them. All further structures that go to make up a program:declaration are composed of strings of symbols. It is always permitted to place one or more spaces between symbols. It is sometimes required to put at least one space between symbols. The criterion is to avoid ambiguity. Comments can often replace required spaces.

.1 Spaces are required in many situations to enable the compiler to detect the end of one symbol and the beginning of the next. Generally, at least one space is required between two symbols of any class except ideograms, but including the quotation:mark. The rule is exhibited in detail in the following table. The rows are labelled with the ending signs of the left symbol of a pair of symbols. The columns are labelled with the beginning signs of the right symbol of a pair. "SR" at the intersection of row and column indicates that at least one space is required between the pair of symbols:

(tab3)

Left symbol ends in	Right symbol starts with: numeral	letter	$	'	.	"
numeral	SR	SR	SR	SR		SR
letter	SR	SR	SR	SR		
$	SR	SR	SR	SR		
'	SR	SR	SR	SR		
.	SR	SR				
"						SR

.2 A comment may occur between symbols. However, it must not occur within a definition nor within any constant, such as a status:constant or a character:constant. A comment may be used instead of the required space between symbols unless use of the comment would cause the occurrence of two quotation:marks in succession. In fact, only the use of a comment can bring about the situation indicated by the lower right corner of the table above. Introduction of a comment between symbols where a space is permitted but not required may then require a space to prevent the comment from interfering with another symbol.

.3 A comment must not be used where the next structure required or permitted by the syntax is a definition. That is, a comment must not follow the define:name or a right:parenthesis in a define:declaration. And a comment must not follow a left:parenthesis or a comma in a definition:invocation. A comment, as defined above, must not occur in a definition delimited by quotation:marks.

<STONE>EQUATIONS.NLS;6, 4-APR-74 06:42 DLS ;

,DefSyn[MonoSpace]=M; ,DefSyn[Slant]=S; ,DefSyn[BoldFace]=B;

(equ1)

1: 233

abbreviation := letter

(equ2)

2: 63

abnormal:directive ::= !ABNORMAL data:name ;

(equ3)

3: 130

absolute:function:call ::= ABS (numeric:formula)

(equ4)

4: 58, 170

actual:define:parameter ::= definition
" definition "

(equ5)

5: 101, 170, 180, 191

```
                                        STOP
    alternate:entrance:name
                                        RETURN
                                                   procedure:name
                                        TEST       control:variable
                                        EXIT       statement:name
        actual:input:parameter  ::=
                                        statement:name
                                        procedure:name
                                        formula
                                        table:name
                                        data:block:name
                                        variable
                                        @ pointer:formula
```

(equ6)

6: 170, 180, 191

actual:output:parameter ::= variable

(equ7)

7: 166,217

allocation:increment ::= number

(equ8)

8: 9, 49, 166, 182, 205, 217

allocation:specifier ::= @ pointer:formula

(equ9)

9: 75, 184

alternate:entrance:declaration ::= ENTER alternate:entrance:name (
formal:input:parameter :
formal:output:parameter)
environmental:specifier
item:description
allocation:specifier
packing:specifier bit:number
= +
constant ;

(equ10)

10: 130

alternate:entrance:function:call ::= ALT (procedure:name)

(equ11)

11: 5, 9, 53, 101, 122, 138, 180, 187, 193

alternate:entrance:name ::= name

(equ12)

12: 159

```
                              +
                              -
                              *
arithmetic:operator ::=
                              /
                              \
                              **
```

```
(equ13)

13:

        assignment:operator  ::=  =


(equ14)

14:    207

                                                  formula
        variable                                  indexed:variable:range
    ;
        indexed:variable:range           =
                                                  format:function:call

                                                  formula
        format:variable                  =
    ;
                                                  indexed:variable:range


(equ15)

15:    159

        attribute:association  ::=  @@  [ description:attibute ]


(equ16)

16:    63

       begin:directive  ::=  !BEGIN   reference   ;


(equ17)

17:    18

        bit:form  ::=  form
```

```
(equ18)

18:    18, 29, 97, 159, 196

                           pattern:constant
                           entry:variable
                           comparison
                           chain:comparison
                           bit:string:function:call
                           shift:function:call
       bit:formula  ::=    bit:form
                           bit:formula  logical:operator
    bit:formula
                           NOT bit:formula
                           bit:formula  &  bit:formula
                           (  bit:formula  )
                           numeric:formula
                           character:formula


(equ19)

19:    9, 182, 205, 217, 218

       bit:number  ::=  number


(equ20)

20:    18, 130

       bit:string:function:call  ::=
       BIT  (  formula  ,  numeric:formula    ,
    numeric:formula    )


(equ21)

21:    247

       bit:variable  ::=
         entry:variable
         BIT  (  named:variable  ,  numeric:formula    ,
    numeric:formula    )
```

(equ22)

22: 217

bits:per:entry ::= number

(equ23)

23: 130

byte:string:function:call ::=
BYTE (character:formula , numeric:formula , numeric:formula)

(equ24)

24: 18

chain:comparison ::= comparison relation:operator formula

(equ25)

25: 26, 32, 46, 62, 100, 112, 120, 137, 213, 234, 240

character ::= sign
system:dependent:character

(equ26)

26: 29, 39

character:constant ::= count ' character '

(equ27)

27: 29

character:form ::= form

(equ28)

28: 29

```
character:format  ::=   count  C
```

(equ29)

29: 18, 23, 93, 94, 97

```
                                  character:constant
                                  character:variable
                                  character:form
     character:formula  ::=       character:function:call
                                  character:formula  &
  character:formula
                                  ( character:formula )
                                  bit:formula
```

(equ30)

30: 29

```
character:function:call  :=  function:call
```

(equ31)

31: 29, 96, 247

```
                                  named:character:variable
     character:variable  ::=      BYTE  (
  named:character:variable  ,  numeric:formula
                                      ,  numeric:formula   )
```

(equ32)

32: 233

```
comment  ::=  "  character  "
```

(equ33)

33: 18, 24

comparison ::= formula relational:operator formula

(equ34)

34: 63

name

compool:name

(name)

compool:directive ::= !COMPOOL

;

(

compool:name)

(equ35)

35: 34

compool:name ::= name

(equ36)

36: 219

declaration

compound:statement ::= BEGIN END

;

statement

(equ37)

37: 38, 97, 238, 241

conditional:formula ::= formula

(equ38)

38: 207

```
conditional:statement ::=
 IF conditional:formula ; controlled:statement

     statement:name :     ELSE controlled:statement
```

(equ39)

39: 9, 42, 182, 205, 233

```
                          numeric:constant
constant:formula ::=      pattern:constant
                          character:constant
```

(equ40)

40: 97

```
constant:formula ::= ( formula )
```

(equ41)

41: 166, 167, 217, 218

```
constant:list ::=
     [ index ]    constant:list:element
          [ index ]
                            constant:list:element
               ,
```

(equ42)

42: 41, 42

```
                                                +
                                          ,     -     constant
   ,
     constant:list:element ::=
                                          count (
constant:list:element )
                                          constant:list:element
```

(equ43)

43: 136, 148

```
    increment:phrase      terminator:phrase

    replacement:phrase
        control:clause  ::=   initial:phase

    increment:phrase

    terminator:phrase     replacement:phrase
```

(equ44)

44: 5, 239

```
                               named:variable
    control:variable  ::=
                               letter:control:variable
```

(equ45)

45: 38, 141

```
        controlled:statement  ::=  statement
```

(equ46)

46: 63

```
        copy:directive  ::=  COPY  character    ;
```

(equ47)

47: 26, 28, 42, 72, 78, 95, 98, 103, 120, 124, 126, 171, 173, 199

```
        count  ::=  number
```

(equ48)

48: 182

```
data:allocator:specifier ::= @
```

(equ49)

49: 49, 51, 75

```
data:block:declaration ::=
                                        environmental:specifier
      BLOCK  data:block:name                                    ;
                                        allocation:specifier

                        simple:item:declaration
                        table:declaration
        BEGIN
   END   ;
                        data:block:declaration
                        independent:overlay:declaration
```

(equ50)

50: 5, 49, 52, 90, 109, 138, 209

```
data:block:name ::= name
```

(equ51)

51: 54

```
                            item:declaration
                            table:declaration
data:declaration ::=
                            data:block:declaration
                            overlay:declaration
```

(equ52)

52: 2, 129, 174, 195, 244

```
                      item:name
    data:name  ::=    table:name
                      data:block:name
```

(equ53)

53: 130

```
procedure:name
    data:size:function:call  ::=  DSIZE  (
)

alternate:entrance:name
```

(equ54)

54: 36, 54, 112, 179

```
                          status:list:declaration
                          form:declaration
                          data:declaration
                          null:declaration
    declaration  ::=      define:declaration
                          name declaration
                          processing:declaration
                          external:declaration
                          BEGIN declaration   END  ;
```

(equ55)

55: 54

```
    define:declaration  ::=  DEFINE define:name  (
formal:define:parameter   )  " definition ";
```

(equ56)

56: 55, 58

define:name ::= name

(equ57)

57: 4, 55

definition ::= sign

(equ58)

58:

definition:invocation ::= define:name (actual:define:parameter)

(equ59)

59: 185

dependent:program:declaration ::= procedure:declaration

(equ60)

60: 15, 69

description:attribute ::= item:name
item:description

(equ61)

61: 166, 217

dimension:list ::= [lower:bound : upper:bound]

(equ62)

62: 207

```
direct:statement  ::=  DIRECT  character  JOVIAL  ;
```

(equ63)

63:

```
                    compool:directive
                    skip:directive
                    begin:directive
                    end:directive
                    trace:directive
                    copy:directive
                    abnormal:directive
                    sets:directive
directive  ::=      uses:directive
                    pointer:directive
                    order:directive
                    recursive:directive
                    time:directive
                    space:directive
                    linkage:directive
                    interference:directive
                    frequency:directive
```

(equ64)

64: 233

```
                         !COMPOOL
                         !SKIP
                         !BEGIN
                         !END
                         !TRACE
                         !COPY
                         !ABNORMAL
                         !SETS
directive:key  ::=       !USES
                         !POINTER
                         !ORDER
                         !RECURSIVE
                         !TIME
                         !SPACE
                         !LINKAGE
                         !INTERFERENCE
                         !FREQUENCY
```

(equ65)

65: 63

end:directive ::= !END ;

(equ66)

66: 217

entries:per:word ::= number

(equ67)

67: 18, 21, 117, 252

entry:variable ::= table:name [index] @ pointer:formula

(equ68)

68: 9, 49, 166, 182, 205, 217

```
                                        program:name
                                   IN   procedure:name
environmental:specifier  ::=            RESERVE

                                   RESERVE
```

(equ69)

69: 159

```
evaluation:control  ::=  @  [  description:attribute  ]
```

(equ70)

70: 207

```
exchange:statement  ::=  variable  ==  variable  ;
```

(equ71)

71: 207

```
exit:statement  ::=  EXIT  statement:name  ;
```

(equ72)

72: 82

```
                         count  D        count  Z

exrad  ::=      +        count  Z        count  D
                -

                         count  Z    *
```

(equ73)

73: 130

```
exrad:function:call  ::=  XRAD  (  numeric:formula  )
```

(equ74)

74: 132

```
exrad:specifier  ::=  number
```

(equ75)

75: 54

```
simple:item:declaration
table:declaration
data:block:declaration
name:declaration
                                          DEF
procedure:declaration
   external:declaration  ::=
alternate:entrance:declaration
                                          REF
simple:item:declaration
 table:declaration
                                                    BEGIN
data:block:declaration    END  ;
name:declaration
procedure:declaration
alternate:entrance:declaration
```

(equ76)

76: 87

```
field:width  ::=  number
```

(equ77)

77: 157

```
                                          number     ,              E    +
     scale    A    +    scale
         fixed:constant  ::=
     -                 -
                                          number     ,    number
```

(equ78)

78: 158

```
                                                                        *
     count   D
                                                         integer:part
                                                                        ,
     fraction:part

                                      +
         fixed:format  ::=                               integer:part     count
     *               R
                                      -

                                                         count   =

     fraction:part
                                                              ,
```

(equ79)

79: 160

fixed:function:call ::= function:call

(equ80)

80: 162

fixed:variable ::= named:variable

(equ81)

81: 157

```
                                                  number   E    +
     scale
                                                                -
          floating:constant  ::=       number    ,        +
     M    +   scale
                                                                    E
     -    scale
                                       number    ,    number
```

(equ82)

82: 158

floating:format ::= significand E exrad R

(equ83)

83: 162

floating:function:call ::= function:call

(equ84)

84: 162

floating:variable ::= named:variable

(equ85)

85: 141

for:clause ::= FOR loop:control ;

(equ86)

86: 17, 27

form ::= form:name (formula)

(equ87)

87: 54

```
form:declaration ::= FORM form:name  B  field:width
    ;                                C
```

(equ88)

88: 86, 87

form:name ::= name

(equ89)

89: 55, 170
formal:define:parameter ::= letter

(equ90)

90: 9, 170, 182

```
                                        statement:name
                                        simple:item:name
formal:input:parameter ::=              procedure:name
                                        table:name
                                        data:block:name
```

(equ91)

91: 9, 170, 182

formal:output:parameter ::= simple:item:name

(equ92)

92: 95

```
                         null:format
                         insert:format
        format  ::=      skip:format
                         character:format
                         pattern:format
    numeric:format
```

(equ93)

93: 14, 130

format:function:call ::= FORMAT character:formula , format:list , procedure:name)

(equ94)

94: 93, 95, 96

format:list ::= character:formula

(equ95)

95: 93, 95, 96

```
                           format
        format:list  ::=
                           count  (  format:list  )
```

(equ96)

96: 14, 102, 247

format:variable ::= FORMAT (character:variable , format:list ,procedure:name)

(equ97)

97: 5, 14, 20, 24, 33, 37, 40, 86, 119, 159, 192, 203, 209, 242, 245

```
                          pointer:formula
                          numeric:formula
                          bit:formula
         formula  ::=     conditional:formula
                          character:formula
                          value:formula
                          numeric:formula
                          constant:formula
```

(equ98)

98: 78

```
                                   count  D        count  Z
         fraction:part  ::=
                                   count  D
```

(equ99)

99: 130

```
         fraction:part:function:call  ::= FRAC  (
     numeric:formula  )
```

(equ100),Grab=n;

100: 65

```
         frequency:directive  ::=  !FREQUENCY  character  ;
```

(equ101)

101: 30, 79, 83, 125

```
                                  intrinsic:function:call
    function:call  ::=            procedure:name     @
pointer:formula
                                  alternate:entrance:name
                                     (  actual:input parameter    )
```

(equ102)

103: 248

```
                                      format:variable
                                      BYTE  (
named:character:variable  ,  numeric:formula

    functional:variable  ::=             ,  numeric:formula    )
                                      BIT  (  named:variable  ,
numeric:formula

                                         ,  numeric:formula    )
```

(equ103)

103: 158

```
    generalized:numeric:formula  ::=    count   N     R
```

(equ104)

104: 207

```
    go:to:statement  ::=  GOTO  statement:name   [  index
]  ;
```

(equ105)

105: 115

```
    high:point  ::=  numeric:formula
```

(equ106)

106: 233

```
                           +
                           -
                           /
                           **
                           \
                           &
                           =
                           ==
                           <
                           >
                           <=
                           >=
    ideogram  ::=          <>
                           .
                           ,
                           :
                           ;
                           !
                           "
                           '
                           (
                           )
                           [
                           ]
                           @
                           @@
```

(equ107)

107: 43

```
    increment:phrase  ::=  BY    numeric:formula
                                 numeric:value:formula
```

(equ108)

108: 49, 168

```
    independent:overlay:declaration  ::=  OVERLAY    [number
  ]
                                                     [
  pattern:constant ]
     independent:overlay:expression  ;
```

(equ109)

109: 111

```
    independent:overlay:element  ::=   spacer
                                       simple:item:name
                                       table:name
                                       data:block:name
                                       (
independent:overlay:expression )
```

(equ110)

110: 108, 109

```
    independent:overlay:expression  ::=
independent:overlay:string
       : independent:overlay:string
```

(equ111)

111: 110

```
    independent:overlay:string  ::=
independent:overlay:element
```

(equ112)

112: 185

```
    independent:program:declaration  ::=  PROGRAM
program:name
        ( character   ) ;  statement
                           declaration
```

(equ113)

113: 41, 67, 104, 237

```
    index  ::=  index:component
```

(equ114)

114: 113, 116

```
index:component  ::=  numeric:formula
```

(equ115)

115: 116

```
index:component:range  ::=  low:point  :  high:point
```

(equ116)

116: 118, 155

```
index:range  ::=  index:component:range
                  index:component
```

(equ117)

117: 150

```
indexed:variable  ::=  table:variable
                       entry:variable
```

(equ118)

118: 14

```
indexed:variable:range  ::=
   item:name  [ index ]   @ pointer:formula
   table:name

 ALL (   item:name     @ pointer:formula   )
         table:name
```

(equ119)

119: 43

```
initial:phrase  ::=  formula
```

(equ120)

120: 92

```
                                    count   S

        insert:format   ::=         count   /   numeral
                                                letter

                                    count   "  character   "
```

(equ121)

121: 182

instruction:allocation:specifier ::= pointer:formula

(equ122)

122: 130

instruction:size:function:call ::= ISIZE (procedure:name)

alternate:entrance:name

(equ123)

123: 157, 175

integer:constant ::= number

(equ124)

124: 158

```
                                count   Z        count   D
                          +
    integer:format  ::=
R
                          -
                                count   D        count   Z
```

(equ125)

125: 160

integer:function:call ::= function:call

(equ126)

126: 78

```
                             count   Z        count   D
    integer:part  ::=
                             count   D
```

(equ127)

127: 130

integer:part:function:call ::= INT (numeric:formula)

(equ128)

128: 162

```
    integer:variable  ::=   named:variable
                            letter:control:variable
```

(equ129)

129: 63

```
    interference:directive  ::=  !INTERFERENCE   data:name
  :  data:name      ;
```

(equ130)

130: 101

```
                                          format:function:call
  byte:string:function:call
                                          bit:string:function:call
  alternate:entrance:function:call
  number:of:entries:function:call
                                          location:function:call
                                          shift:function:call
                                          absolute:function:call
  words:per:entry:function:call
      intrinsic:function:call  ::=        exrad
  significand:function:call
                                          signum:function:call
                                          size:function:call
                                          type:function:call
  fraction:part:function:call
  integer:part:function:call
  instruction:size:function:call
                                          data:size:function:call
```

(equ131)

131: 51

```
                                   simple:item:declaration
      item:declaration  ::=        ordinary:table:item:declaration
                                   specified:table:item:declaration
```

(equ132)

132: 9, 60, 166, 167, 182, 205, 217, 218

```
item:description ::=

C   size:specifier

F   , R    significand:specifier      , exrad:specifier

                                          status:list
 S                                       status:list:name
    , R    size:specifier           +
precision:specifier
 U                                ,   -
```

(equ133)

133: 52, 60, 118, 167, 187, 205, 218, 229, 237,

item:name ::= name

(equ134)

134: 1, 89, 120, 135, 136, 145, 197, 221

```
                A
                B
                C
                D
                E
                F
                G
                H
                I
                J
                K
                L
letter  ::=     M
                N
                O
                P
                Q
                R
                S
                T
                U
                V
                W
                X
                Y
                Z
```

(equ135)

135: 44, 128, 233, 248

letter:control:variable ::= letter

(equ136)

136: 140

letter:loop:control ::= letter (control:clause)

```
(equ137)

137:   63

        linkage:directive  ::=  !LINKAGE   character     ;


(equ138)

138:   130

                                                    statement:name
                                                    named:variable
        location:function:call  ::=  LOC (        table:name
    )
                                                    data:block:name
                                                    procedure:name

    alternate:entrance:name


(equ139)

139:   130

                                        AND
        logical:operator  ::=           OR
                                        EQV
                                        XOR


(equ140)

140:   85

        logical:operator  ::=    named:loop:control
                                 letter:loop:control


(equ141)

141:   207
        loop:statement  ::=  for:clause contolled:statement
```

```
(equ142)

142:   115

        low:point  ::=  numeric:formula


(equ143)

143:   61

        lower:bound  ::=   number
                           simple:item:name


(equ144)

144:   197

                          +       plus:sign
                          -       minus:sign
                          *       asterisk
                          /       slash
                          \       back:slash
                          &       ampersand
                          >       greater:than:sign
                          <       less:than:sign
                          =       equals:sign
                          @       at:sign
        mark  ::=         .       decimal:point
                          :       colon
                          ,       comma
                          ;       semicolon
                                  space
                          (       left:parenthesis, parenthesis
                          )       right:parenthesis, parenthesis
                          [       left:backet, bracket
                          ]       right:bracket, bracket
                          '       prime
                          "       quotation:mark
                          $       dollar:sign
                          !       exclamation:point
```

(equ145)

145: 11, 34, 35, 50, 56, 88, 133, 183, 186, 206, 220, 221, 225, 233, 236, 241

```
                                    letter
name  ::=     letter               numeral
                 $                    $
                                      ,
```

(equ146)

146: 54, 75

```
name:declaration  ::=  NAME   statement:name      ;
                                procedure:name
```

(equ147)

147: 31, 102

named:character:variable ::= named:variable

(equ148)

148: 140

named:loop:control ::= named:variable (control:clause)

(equ149)

149: 219

named:statement ::= statement:name : statement

(equ150)

150: 21, 44, 80, 84, 102, 128, 138, 147, 148

```
named:variable  ::=  simple:variable
                     indexed:variable
```

(equ151)

151: 54, 164, 215

```
                               NULL   ;
null:declaration  ::=
                               BEGIN  END  ;
```

(equ152)

152: 92

```
null:format  ::=
```

(equ153)

153: 219

```
                               NULL   ;
null:declaration  ::=
                               BEGIN  END  ;
```

(equ154)

154: 7, 19, 22, 47, 66, 74, 76, 77, 81, 108, 123, 143, 169, 177, 189, 194, 201, 210, 214, 223, 232, 233, 243, 249, 250

```
number  ::=  numeral
```

(equ155)

155: 130

```
    number:of:entries:function:call  ::=  NENT ( table:name
  [index:range]   )
```

(equ156)

156: 120, 145, 154, 197

```
                        0
                        1
                        2
                        3
    numeral  ::=        4
                        5
                        6
                        7
                        8
                        9
```

(equ157)

157: 39, 159

```
                               integer:constant
                               fixed:constant
    numeric:constant  ::=      floating:constant
                               status:constant
                               qualified:status:constant
```

(equ158)

158: 92

```
                            generalized:numeric:format
    numeric:format  ::=     integer:format
                            fixed:format
                            floating:format
```

(equ159)

159: 3, 18, 20, 21, 23, 31, 73, 97, 99, 102, 105, 107, 114, 127, 142, 159, 161, 175, 196, 198, 200, 202, 232

```
                                      numeric:constant
                                      numeric:variable
                                      numeric:function:call
                                        +  numeric:formula
                                        -
                                      numeric:formula
    arithmetic:operator
       numeric:formula  ::=
                                          evaluation:control
    numeric:formula

                                      formula    evaluation:control
                                      attribute:association

                                      ( numeric:formula
                                      bit:formula
```

(equ160)

160: 159

```
       numeric:variable  ::=   integer:variable
                               fixed:variable
                               floating:variable
```

(equ161)

161: 97, 107

```
       numeric:value:formula  ::=  [ numeric:formula ]
```

(equ162)

162: 159, 176, 247

```
       numeric:variable  ::=   integer:variable
                               fixed:variable
                               floating:variable
```

(equ163)

163: 63

```
        order:directive  ::=  !ORDER  ;
```

(equ164)

164: 165

```
                                              null:declaration

    ordinary:table:item:declaration
        ordinary:table:body  ::=
                                              BEGIN
    ordinary:table:item:declaration             END   ;

    subordinate:overlay:declaration
```

(equ165)

165: 235

```
        ordinary:table:declaration  ::=  ordinary:table:heading
    ordinary:table:body
```

(equ166)

166: 165

```
        ordinary:table:heading  ::=
         TABLE  table:name    environmental:specifier
                                     allocation:specifier

         : allocation:increment   dimension:list

         structure:specifier    packing:specifier

         item:description    = constant:list   ;
```

(equ167)

167: 131, 164

ordinary:table:item:declaration ::=
ITEM item:name item:description packing:specifier
= constant:list ;

(equ168)

168: 51

overlay:declaration ::=
independent:overlay:declaration

subordinate:overlay:declaration

(equ169)

169: 9, 166, 167, 182, 205, 217, 218

```
                              N
packing:specifier  ::=        M   number
                              D
```

(equ170)

170:

```
                        actual:define:parameter
                        formal:define:parameter
parameter  ::=          actual:input:parameter
                        formal:input:parameter
                        actual:output:parameter
                        formal:output:parameter
```

(equ171)

171: 18, 39, 108

```
                                   1
                                   2
        pattern:constant  ::=      3    B    count    '
    pattern:digit    '
                                   4
                                   5
```

(equ172)

172: 171

pattern:digit ::=

pattern	pattern:digit	order
0 0 0 0 0	0	
0 0 0 0 1	1	1
0 0 0 1 0	2	
0 0 0 1 1	3	
0 0 1 0 0	4	
0 0 1 0 1	5	
0 0 1 1 0	6	
0 0 1 1 1	7	3
0 1 0 0 0	8	
0 1 0 0 1	9	
0 1 0 1 0	A	
0 1 0 1 1	B	
0 1 1 0 0	C	
0 1 1 0 1	D	
0 1 1 1 0	E	
0 1 1 1 1	F	4
1 0 0 0 0	G	
1 0 0 0 1	H	
1 0 0 1 0	I	
1 0 0 1 1	J	
1 0 1 0 0	K	
1 0 1 0 1	L	
1 0 1 1 0	M	
1 0 1 1 1	N	
1 1 0 0 0	O	
1 1 0 0 1	P	
1 1 0 1 0	Q	
1 1 0 1 1	R	
1 1 1 0 0	S	
1 1 1 0 1	T	
1 1 1 1 0	U	
1 1 1 1 1	V	5

(equ173)

173: 92

```
                              1
                              2
pattern:format  ::=           3    B     count   P
                              4
                              5
```

(equ174)

174: 63

pointer:directive ::= ¦POINTER pointer:formula : data:name ;

(equ175)

175: 5, 8, 67, 97, 101, 118, 121, 174, 180, 208, 237

```
                          integer:constant
pointer:formula  ::=      simple:integer:variable
                          ( numeric:formula )
```

(equ176)

176: 247

pointer:variable ::= numeric:variable

(equ177)

177: 132

precision:specifier ::= number

(equ178)

178: 221, 233

```
                                ABS
                                ALL
                                ALT             NENT
                                AND             NOT
                                BEGIN           NULL
                                FIT             NWDSEN
                                BLOCK           OR
                                BY              OVERLAY
                                BYTE            PROC
                                DEF             PROGRAM
                                DEFINE          REF
                                DIRECT          REMQUO
                                DSIZE           RESERVE
                                ELSE            RETURN
                                END             SHIFT
        primitive  ::=          ENTER           SIG
                                EQV             SIGNED
                                EXIT            SIGNUM
                                FOR             SIZE
                                FORM            STATUS
                                FORMAT          STOP
                                FRAC            SWITCH
                                GOTO            TABLE
                                IF              TEST
                                IN              THEN
                                INT             TYPE
                                ISIZE           UNTIL
                                ITEM            WHILE
                                JOVIAL          XOR
                                LOC             XRAD
                                NAME            ZAP
```

(equ179)

179: 181

```
        procedure;body  ::=     declaration
                                statement
```

(equ180)

180: 207

remquo:procedure:call:statement

procedure:name

@ pointer:formula

alternate:entrance:name
procedure:call:statement ::=

(

actual:input:parameter)

(

actual:input:parameter : ;

actual:output:parameter)

(equ181)

181: 59, 75, 184

procedure:declaration ::= procedure:heading
procedure:body

(equ182)

182: 181
procedure:heading ::=
PROC procedure:name environmental:specifier
data:allocation:specifier

: instruction:allocation:specifier

(formal:input:parameter :
formal:output:parameter)

environmental:specifier item:description
allocation:specifier

packing:specifier [bit:number]

= + constant ;
-

(equ183)

183: 5, 10, 53, 68, 90, 96, 101, 122, 138, 146, 180, 182, 167, 193

procedure:name ::= name

(equ184)

184: 54

processing:declaration ::= program:declaration
procedure:declaration
alternate:entrance:declaration

(equ185)

185: 184

program:declaraton ::=
independent:program:declaration
dependent:program:declaration

(equ186)

186: 68, 112

program:name ::= name

(equ187)

187: 157

qualified:status:constant ::= V(status:list:name
item:name
table:name
procedure:name
alternate:entrance:name
: status)

```
(equ188)

188:   63

        recursive:directive  ::=  !RECURSIVE  ;


(equ189)

189:   16, 211

        reference  ::=  number


(equ190)

190:   24, 33, 246

                                        <      less than
                                        =      equal
        relational:operator  ::=        >      greater than
                                        >=     greater than or
    equal, not less than
                                        <>     less than or greater
    than, not equal
                                        <=     less than or equal,
    not greater than


(equ191)

191:   180

        remquo:procedure:call:statement  ::=
         REMQUO ( actual:input:parameter ,
         actual:input:parameter
            : actual:output:parameter , actual:output:parameter
          ) ;


(equ192)

192:   43

        replacement:phrase  ::=  THEN    formula
                                          value:formula
```

(equ193)

193: 207

```
return:statement ::= RETURN   procedure:name
    ;                         alternate:entrance:name
```

(equ194)

194: 77, 81

```
scale ::= number
```

(equ195)

195: 63

```
sets:directive ::= !SETS  data:name    ;
```

(equ196)

196: 18, 130

```
shift:function:call ::= SHIFT ( bit:formula ,
numeric:formula )
```

(equ197)

197: 25, 57, 234

```
               letter
sign ::=       numeral
               mark
```

(equ198)

198: 18, 130

```
signed:function:call ::= SIGNED ( numeric:formula )
```

(equ199)

199: 82

```
                                          count  D  .  count  D
    count  Z
                                          count  D  .  count  D

                                          count  D  .
         significand ::=   +
                           -
                                          count  D  *  count  D

                                          count  *      count  D

                                          count  D      count  *
```

(equ200)

200: 130

significand:function:call ::= SIG (numeric:formula)

(equ201)

201: 132

significand:specifier ::= number

(equ202)

202: 130

signum:function:call ::= SIGNUM (numeric:formula)

(equ203)

203: 207

simple:assignment:statement ::= variable = formula ;

(equ204)

204: 175

simple:integer:variable ::= simple:variable

(equ205)

205: 49, 75, 131

simple:item:declaration ::=
ITEM item:name environmental:specifier
allocation:specifier

item:description packing:specifier

[bit:number] = + constant ;
-

(equ206)

206: 90, 91, 109, 143, 208, 243

simple:item:name ::= name

(equ207)

207: 219

```
                                  simple:assignment:statement
                                  assignment:statement
                                  exchange:statement
                                  go:to:statement
                                  exit:statement
                                  test:statement
    simple:statement  ::=         return:statement
                                  zap:statement
                                  stop:statement
                                  loop:statement
                                  conditional:statement
                                  switch:statement
                                  procedure:call:statement
                                  direct:statement
```

(equ208)

208: 150, 204

simple:variable ::= simple:item:name @ pointer:formula

(equ209)

209: 130

```
                                           formula
    size:function:call  ::=  SIZE  (                      )
                                           data:block:name
```

(equ210)

210: 132

size:specifier ::= number

```
(equ211)

211:   63

       skip:directive  ::=  !SKIP    reference    ;


(equ212)

212:   92

       skip:format  ::=  X


(equ213)

213:   63

       space:directive  ::=  !SPACE    character    ;


(equ214)

214:   109

       spacer  ::=  number


(equ215)

215:   216

                                          null:declaration
       specified:table:body  ::=
    specified:table:item:declaration
                                          BEGIN
    specified:table:item:declaration       END    ;


(equ216)

216:   235

       specified:table:declaration  ::=
    specified:table:heading   specified:table:body
```

```
(equ217)

217:   216

       environmental:specifier
           specified:table:heading  ::=  TABLE table:name

       allocation:specifier

                : allocation:increment    dimension:list

                structure:specifier

                words:per:entry
                bits:per:entry    bit:number    entries:per:word

                packingspecifier    item:description
                packing:specifier

                [ bit:number    , word:number    ]    =
                constant:list    ;


(equ218)

218:   131, 215

           specified:table:item:declaration  ::=  ITEM item:name
       item:description

                packing:specifier    [ bit:number
                , word:number   ]   = constant   ;


(equ219)

219:   36, 45, 112, 149, 179, 232

                             null:statement
           statement  ::=    simple:statement
                             compound:statement
                             named:statement
```

(equ220)

220: 5, 38, 71, 90, 104, 138, 146, 149, 232

statement:name ::= name

(equ221)

221: 187, 222, 233

```
status  ::=    primitive
               name
               letter
```

(equ222)

222: 157, 223

status:constant ::= V(status)

(equ223)

223: 132, 224

```
status:list  ::=     [  +  number ]  status:constant
                        -

          [  +  number ] status:constant
             -
```

(equ224)

224: 54

status:list:declaration ::= STATUS status:list:name status:list ;

(equ225)

225: 132, 187, 224

status:list:name ::= name

(equ226)

226: 207

```
stop:statement ::= STOP ;
```

(equ227)

227: 166, 217

```
                                  P
structure:specifier ::=
                                  T
```

(equ228)

228: 164, 168

```
    subordinate:overlay:declaration ::= OVERLAY
subordinate:overlay:expression ;
```

(equ229)

229: 231

```
                                              item:name
    subordinate:overlay:element ::=
                                              (
subordinate:overlay:expression )
```

(equ230)

230: 228, 229

```
    subordinate:overlay:expression ::=

        subordinate:overlay:string :
        subordinate:overlay:string
```

(equ231)

231: 230

```
    subordinate:overlay:string  ::=
subordinate:overlay:element
```

(equ232)

232: 207

```
    switch:statement  ::=  SWITCH numeric:formula ;
statement:name :

            BEGIN    [  +  number ]    statement   ,
                        -

            END   ;
```

(equ233)

233:

```
                         primitive
                         ideogram
                         name
                         letter:control:variable
    symbol  ::=          abbreviation
                         number
                         constant
                         comment
                         directive:key
                         status
```

(equ234)

234: 25

system:dependent:character
Most computer systems can read and write more characters than are encompassed in the set of JOVIAL "sign , The entire set that can be handled is know as the set of "characters , The "characters that are not "signs are known as "system:dependent:characters ,

(equ235)

235: 49, 51, 75

table:declaration ::= ordinary:table:declaration
specified:table:declaration

(equ236)

236: 5, 52, 67, 90, 109, 118, 138, 155, 166, 187, 217, 251, 252

table:name ::= name

(equ237)

237: 117
table:variable ::= item:name [index] @
pointer:formula

(equ238)

238: 43

terminator:phrase ::= WHILE conditional:formula
UNTIL
value:terminator

(equ239)

239: 207

test:statement ::= TEST character ;

(equ240)

240: 63

time:directive ::= !TIME character ;

```
(equ241)

241:   63

      trace:directive  ::=  !TRACE   ( conditional:formula )
   name    ;


(equ242)

242:   130

      type:function:call  ::=  TYPE  ( formula )


(equ243)

243:   61

                             number
      upper:bound  ::=
                             simple:item:name


(equ244)

244:   63

      uses:directive  ::=  !USES  data:name     ;


(equ245)

245:   97, 192, 246

      value:formula  ::=  [ formula ]


(equ246)

246:   238
      value:terminator  ::=

            WHILE    value:formula  relational:operator
            variable
            UNTIL    variable  relational:operator
            value:formula
```

(equ247)

247: 5, 6, 14, 70, 203, 246

```
                          pointer:variable
                          numeric:variable
        variable  ::=     bit:variable
                          character:variable
                          format:variable
```

(equ248)

248: 5, 6, 14, 70, 203, 246

```
                          named:variable
        variable  ::=     letter:control:variable
                          functional:variable
```

(equ249)

249: 217, 218

```
        word:number  ::=  number
```

(equ250)

250: 217

```
        words:per:entry  ::=  number
```

(equ251)

251: 130

```
        words:per:entry:function:call  ::=  NWDSEN  ( table:name
  )
```

(equ252)

252: 207

```
                              table:name
zap:statement  ::=  ZAP                              ;
                              entry:variable
```

,DefSyn[MonoSpace]=M; ,DefSyn[Slant]=S; ,DefSyn[BoldFace]=B;

```
(equ1)

1:      233

          abbreviation  :=  letter


(equ2)

2:      63

          abnormal:directive  ::=  !ABNORMAL  data:name      ;


(equ3)

3:      130

          absolute:function:call  ::=  ABS  (  numeric:formula  )


(equ4)

4:    58, 170

                                                  definition
      actual:define:parameter  ::=
                                                " definition "


(equ5)

5:      101, 170, 180, 191

                                              STOP

      alternate:entrance:name
                                              RETURN
                                                      procedure:name
                                              TEST    control:variable
                                              EXIT    statement:name
          actual:input:parameter  ::=
                                              statement:name
                                              procedure:name
                                              formula
                                              table:name
                                              data:block:name
                                              variable
```

@ pointer:formula

(equ6)

6: 170, 180, 191

actual:output:parameter ::= variable

(equ7)

7: 166,217

allocation:increment ::= number

(equ8)

8: 9, 49, 166, 182, 205, 217

allocation:specifier ::= @ pointer:formula

(equ9)

9: 75, 184

alternate:entrance:declaration ::= ENTER
alternate:entrance:name (
formal:input:parameter :
formal:output:parameter)
environmental:specifier
item:description
allocation:specifier
packing:specifier bit:number = +
constant ;

```
(equ10)

10:    130

          alternate:entrance:function:call  ::=  ALT  (
      procedure:name    )


(equ11)

11:    5, 9, 53, 101, 122, 138, 180, 187, 193

          alternate:entrance:name  ::=  name


(equ12)

12:    159

                                             +
                                             -
                                             *
          arithmetic:operator  ::=
                                             /
                                             \
                                             **


(equ13)

13:

          assignment:operator  ::=  =


(equ14)

14:    207

                                                  formula
          variable                                indexed:variable:range
      ;
          indexed:variable:range        =
                                                  format:function:call

                                                  formula
          format:variable               =
      ;
```

indexed:variable:range

(equ15)

15: 159

attribute:association ::= @@ [description:attibute]

(equ16)

16: 63

begin:directive ::= !BEGIN reference ;

(equ17)

17: 18

bit:form ::= form

(equ18)

18: 18, 29, 97, 159, 196

```
                              pattern:constant
                              entry:variable
                              comparison
                              chain:comparison
                              bit:string:function:call
                              shift:function:call
      bit:formula  ::=        bit:form
                              bit:formula  logical:operator
   bit:formula
                              NOT bit:formula
                              bit:formula  &  bit:formula
                              (  bit:formula  )
                              numeric:formula
                              character:formula
```

(equ19)

19: 9, 182, 205, 217, 218

bit:number ::= number

(equ20)

20: 18, 130

bit:string:function:call ::=
BIT (formula , numeric:formula ,
numeric:formula)

(equ21)

21: 247

bit:variable ::=
entry:variable
BIT (named:variable , numeric:formula ,
numeric:formula)

(equ22)

22: 217

bits:per:entry ::= number

(equ23)

23: 130

byte:string:function:call ::=
BYTE (character:formula , numeric:formula ,
numeric:formula)

(equ24)

24: 18

bh'hn:bnlp'rhson ::= comparison relation:operator formula

(equ25)

25: 26, 32, 46, 62, 100, 112, 120, 137, 213, 234, 240

```
character ::=  sign
               system:dependent:character
```

(equ26)

26: 29, 39

character:constant ::= count ' character '

(equ27)

27: 29

character:form ::= form

(equ28)

28: 29

character:format ::= count C

(equ29)

29: 18, 23, 93, 94, 97

```
                             character:constant
                             character:variable
                             character:form
character:formula ::=        character:function:call
                             character:formula &
     character:formula
                             ( character:formula )
                             bit:formula
```

(equ30)

30: 29

```
character:function:call  := function:call
```

(equ31)

31: 29, 96, 247

```
                                    named:character:variable
    character:variable  ::=    BYTE  (
named:character:variable  ,  numeric:formula
                                   ,  numeric:formula    )
```

(equ32)

32: 233

```
    comment  ::=  "  character  "
```

(equ33)

33: 18, 24

```
    comparison  ::=  formula  relational:operator  formula
```

(equ34)

34: 63

```
name
                                              compool:name
(  name  )
    compool:directive  ::=  !COMPOOL
;
                                                     (
compool:name  )
```

(equ35)

35: 34

```
compool:name  ::=  name
```

(equ36)

36: 219

```
                                     declaration
compound:statement  ::=  BEGIN                         END
   ;
                                     statement
```

(equ37)

37: 38, 97, 238, 241

```
conditional:formula  ::=  formula
```

(equ38)

38: 207

```
conditional:statement  ::=
 IF  conditional:formula  ;  controlled:statement

    statement:name  :     ELSE  controlled:statement
```

(equ39)

39: 9, 42, 182, 205, 233

```
                           numeric:constant
constant:formula  ::=      pattern:constant
                           character:constant
```

(equ40)

40: 97

constant:formula ::= (formula)

(equ41)

41: 166, 167, 217, 218

constant:list ::=
[index] constant:list:element
[index]
constant:list:element
,

(equ42)

42: 41, 42

+
, - constant
,
constant:list:element ::=
count (
constant:list:element)
constant:list:element

(equ43)

43: 136, 148

increment:phrase terminator:phrase

replacement:phrase
control:clause ::= initial:phase

increment:phrase

terminator:phrase replacement:phrase

(equ44)

44: 5, 239

control:variable ::= named:variable
letter:control:variable

(equ45)

45: 38, 141

controlled:statement ::= statement

(equ46)

46: 63

copy:directive ::= COPY character ;

(equ47)

47: 26, 28, 42, 72, 78, 95, 98, 103, 120, 124, 126, 171, 173, 199

count ::= number

(equ48)

48: 182

data:allocator:specifier ::= @

(equ49)

49: 49, 51, 75

```
    data:block:declaration  ::=
                                            environmental:specifier
          BLOCK  data:block:name                                    ;
                                            allocation:specifier

                          simple:item:declaration
                          table:declaration
             BEGIN
      END     ;
                          data:block:declaration
                          independent:overlay:declaration
```

(equ50)

50: 5, 49, 52, 90, 109, 138, 209

```
    data:block:name  ::=  name
```

(equ51)

51: 54

```
                               item:declaration
                               table:declaration
    data:declaration  ::=
                               data:block:declaration
                               overlay:declaration
```

(equ52)

52: 2, 129, 174, 195, 244

```
                         item:name
    data:name  ::=       table:name
                         data:block:name
```

```
(equ53)

53:    130

     procedure:name
        data:size:function:call  ::=  DSIZE  (
     )

     alternate:entrance:name


(equ54)

54:    36, 54, 112, 179

                                 status:list:declaration
                                 form:declaration
                                 data:declaration
                                 null:declaration
        declaration  ::=         define:declaration
                                 name declaration
                                 processing:declaration
                                 external:declaration
                                 BEGIN declaration   END  ;


(equ55)

55:    54

        define:declaration  ::=  DEFINE define:name  (
     formal:define:parameter   )  " definition ";


(equ56)

56:    55, 58

        define:name  ::=  name


(equ57)

57:    4, 55

        definition  ::=  sign
```

(equ58)

58:

definition:invocation ::= define:name (actual:define:parameter)

(equ59)

59: 185

dependent:program:declaration ::= procedure:declaration

(equ60)

60: 15, 69

description:attribute ::= item:name
item:description

(equ61)

61: 166, 217

dimension:list ::= [lower:bound : upper:bound]

(equ62)

62: 207

direct:statement ::= DIRECT character JOVIAL ;

(equ63)

63:

```
                    compool:directive
                    skip:directive
                    begin:directive
                    end:directive
                    trace:directive
                    copy:directive
                    abnormal:directive
                    sets:directive
directive  ::=      uses:directive
                    pointer:directive
                    order:directive
                    recursive:directive
                    time:directive
                    space:directive
                    linkage:directive
                    interference:directive
                    frequency:directive
```

(equ64)

64:    233

```
                         !COMPOOL
                         !SKIP
                         !BEGIN
                         !END
                         !TRACE
                         !COPY
                         !ABNORMAL
                         !SETS
directive:key  ::=       !USES
                         !POINTER
                         !ORDER
                         !RECURSIVE
                         !TIME
                         !SPACE
                         !LINKAGE
                         !INTERFERENCE
                         !FREQUENCY
```

(equ65)

65: 63

```
end:directive ::= !END ;
```

(equ66)

66: 217

```
entries:per:word ::= number
```

(equ67)

67: 18, 21, 117, 252

```
entry:variable ::= table:name [ index ] @
pointer:formula
```

(equ68)

68: 9, 49, 166, 182, 205, 217

```
                                        program:name
                                   IN   procedure:name
environmental:specifier ::=             RESERVE

                                   RESERVE
```

(equ69)

69: 159

```
evaluation:control ::= @ [ description:attribute ]
```

(equ70)

70: 207

```
exchange:statement ::= variable == variable ;
```

(equ71)

71: 207

exit:statement ::= EXIT statement:name ;

(equ72)

72: 82

```
                                   count  D        count  Z

      exrad  ::=     +
                     -             count  Z        count  D

                                   count  Z     *
```

(equ73)

73: 130

exrad:function:call ::= XRAD (numeric:formula)

(equ74)

74: 132

exrad:specifier ::= number

(equ75)

75: 54

```
    simple:item:declaration

    table:declaration

    data:block:declaration

    name:declaration
                                                DEF
    procedure:declaration
        external:declaration  ::=
    alternate:entrance:declaration
                                                REF
    simple:item:declaration
     table:declaration
                                                      BEGIN
    data:block:declaration     END  ;

    name:declaration

    procedure:declaration

    alternate:entrance:declaration
```

(equ76)

76: 87

```
        field:width  ::=  number
```

(equ77)

77: 157

```
                                    number   ,               E    +
    scale    A    +    scale
        fixed:constant  ::=
    -                  -
                                    number   ,    number
```

(equ78)

78: 158

```
                                                         *
     count   D
                                          integer:part
                                                         .
     fraction:part

        fixed:format  ::=        +        integer:part    count
     *             R
                                 -

                                          count   =

     fraction:part
                                          .
```

(equ79)

79: 160

fixed:function:call ::= function:call

(equ80)

80: 162

fixed:variable ::= named:variable

(equ81)

81: 157

```
                                            number   E    +
     scale
                                                          -
        floating:constant  ::=      number   .           +
     M    +    scale
                                                              E
     -    scale
                                    number   .    number
```

(equ82)

82: 158

floating:format ::= significand E exrad R

(equ83)

83: 162

floating:function:call ::= function:call

(equ84)

84: 162

floating:variable ::= named:variable

(equ85)

85: 141

for:clause ::= FOR loop:control ;

(equ86)

86: 17, 27

form ::= form:name (formula)

(equ87)

87: 54

form:declaration ::= FORM form:name B field:width ;
C

(equ88)

88: 86, 87

```
form:name  ::=  name
```

(equ89)

89: 55, 170

```
formal:define:parameter  ::=  letter
```

(equ90)

90: 9, 170, 182

```
                                  statement:name
                                  simple:item:name
formal:input:parameter  ::=      procedure:name
                                  table:name
                                  data:block:name
```

(equ91)

91: 9, 170, 182

```
formal:output:parameter  ::=  simple:item:name
```

(equ92)

92: 95

```
                  null:format
                  insert:format
format  ::=       skip:format
                  character:format
                  pattern:format
numeric:format
```

(equ93)

93: 14, 130

format:function:call ::= FORMAT character:formula , format:list , procedure:name)

(equ94)

94: 93, 95, 96

format:list ::= character:formula

(equ95)

95: 93, 95, 96

```
                           format
format:list  ::=
                           count  (  format:list  )
```

(equ96)

96: 14, 102, 247

format:variable ::= FORMAT (character:variable , format:list ,procedure:name)

(equ97)

97: 5, 14, 20, 24, 33, 37, 40, 86, 119, 159, 192, 203, 209, 242, 245

```
                  pointer:formula
                  numeric:formula
                  bit:formula
formula  ::=      conditional:formula
                  character:formula
                  value:formula
                  numeric:formula
                  constant:formula
```

(equ98)

98: 78

```
                                    count   D        count   Z
        fraction:part  ::=
                                    count   D
```

(equ99)

99: 130

```
        fraction:part:function:call  ::= FRAC  (
     numeric:formula  )
```

(equ100),Grab=n;

100: 65

```
        frequency:directive  ::=  !FREQUENCY  character   ;
```

(equ101)

101: 30, 79, 83, 125

```
                                 intrinsic:function:call
        function:call  ::=       procedure:name    @
     pointer:formula
                                 alternate:entrance:name
                                    (  actual:input parameter    )
```

(equ102)

103: 248

```
                                        format:variable
                                        BYTE  (
    named:character:variable  ,  numeric:formula

       functional:variable  ::=         ,  numeric:formula    )
                                        BIT  (  named:variable  ,
    numeric:formula

                                           ,  numeric:formula    )
```

(equ103)

103: 158

generalized:numeric:formula ::= count N R

(equ104)

104: 207

go:to:statement ::= GOTO statement:name [index] ;

(equ105)

105: 115

high:point ::= numeric:formula

(equ106)

106: 233

```
                           +
                           -
                           /
                           **
                           \
                           &
                           =
                           ==
                           <
                           >
                           <=
                           >=
        ideogram  ::=      <>
                           .
                           ,
                           :
                           ;
                           !
                           "
                           '
                           (
                           )
                           [
                           ]
                           @
                           @@
```

(equ107)

107: 43

```
        increment;phrase  ::=  BY    numeric:formula
                                     numeric;value:formula
```

(equ108)

108: 49, 168

```
      independent:overlay;declaration  ::=  OVERLAY    [number
   ]
                                                        [
   pattern:constant ]
        independent:overlay;expression  ;
```

(equ109)

109: 111

```
    independent:overlay:element  ::=   spacer
                                       simple:item:name
                                       table:name
                                       data:block:name
                                       (
independent:overlay:expression )
```

(equ110)

110: 108, 109

```
    independent:overlay:expression  ::=
independent:overlay:string
      : independent:overlay:string
```

(equ111)

111: 110

```
    independent:overlay:string  ::=
independent:overlay:element
```

(equ112)

112: 185

```
    independent:program:declaration  ::=  PROGRAM
program:name
       ( character   ) ;  statement
                          declaration
```

(equ113)

113: 41, 67, 104, 237

```
    index  ::=  index:component
```

(equi14)

114: 113, 116

```
index:component ::= numeric:formula
```

(equi15)

115: 116

```
index:component:range ::= low:point : high:point
```

(equi16)

116: 118, 155

```
index:range ::= index:component:range
                index:component
```

(equi17)

117: 150

```
indexed:variable ::= table:variable
                     entry:variable
```

(equi18)

118: 14

```
indexed:variable:range ::=
   item:name [ index ]   @ pointer:formula
   table:name

 ALL (   item:name     @ pointer:formula   )
           table:name
```

(equi19)

119: 43

```
initial:phrase ::= formula
```

(equ120)

120: 92

```
                         count   S
    insert:format  ::=   count   /   numeral
                                     letter
                         count   "  character   "
```

(equ121)

121: 182

instruction:allocation:specifier ::= pointer:formula

(equ122)

122: 130

instruction:size:function:call ::= ISIZE (procedure:name)

alternate:entrance:name

(equ123)

123: 157, 175

integer:constant ::= number

(equ124)

124: 158

```
                                  count   Z       count   D
                        +
       integer:format  ::=
   R
                        -
                                  count   D       count   Z
```

(equ125)

125: 160

integer:function:call ::= function:call

(equ126)

126: 78

```
                            count   Z      count   D
       integer:part  ::=
                            count   D
```

(equ127)

127: 130

integer:part:function:call ::= INT (numeric:formula)

(equ128)

128: 162

```
       integer:variable  ::=   named:variable
                               letter:control:variable
```

(equ129)

129: 63

interference:directive ::= !INTERFERENCE data:name : data:name ;

(equ130)

130: 101

```
                                        format:function:call
  byte:string:function:call
                                        bit:string:function:call
  alternate:entrance:function:call
  number:of:entries:function:call
                                        location:function:call
                                        shift:function:call
                                        absolute:function:call
  words:per:entry:function:call
     intrinsic:function:call  ::=       exrad
  significand:function:call
                                        signum:function:call
                                        size:function:call
                                        type:function:call
  fraction:part:function:call
  integer:part:function:call
  instruction:size:function:call
                                        data:size:function:call
```

(equ131)

131: 51

```
                               simple:item:declaration
     item:declaration  ::=     ordinary:table:item:declaration
                               specified:table:item:declaration
```

(equ132)

132: 9, 60, 166, 167, 182, 205, 217, 218

```
        item:description  ::=

         C   size:specifier

         F   , R    significand:specifier     , exrad:specifier

                                                 status:list
          S                                     status:list:name
               , R    size:specifier           +
        precision:specifier
          U                                ,   -
```

(equ133)

133: 52, 60, 118, 167, 187, 205, 218, 229, 237,

item:name ::= name

(equ134)

134: 1, 89, 120, 135, 136, 145, 197, 221

```
                    A
                    B
                    C
                    D
                    E
                    F
                    G
                    H
                    I
                    J
                    K
                    L
letter  ::=         M
                    N
                    O
                    P
                    Q
                    R
                    S
                    T
                    U
                    V
                    W
                    X
                    Y
                    Z
```

(equ135)

135: 44, 128, 233, 248

letter:control:variable ::= letter

(equ136)

136: 140

letter:loop:control ::= letter (control:clause)

(equ137)

137: 63

```
linkage:directive ::= !LINKAGE character ;
```

(equ138)

138: 130

```
                                                  statement:name
                                                  named:variable
location:function:call ::= LOC (          table:name
    )
                                                  data:block:name
                                                  procedure:name

    alternate:entrance:name
```

(equ139)

139: 130

```
                             AND
logical:operator ::=         OR
                             EQV
                             XOR
```

(equ140)

140: 85

```
logical:operator ::=         named:loop:control
                             letter:loop:control
```

(equ141)

141: 207

```
loop:statement ::= for:clause contolled:statement
```

(equ142)

142: 115

```
low:point  ::=  numeric:formula
```

(equ143)

143: 61

```
lower:bound  ::=   number
                   simple:item:name
```

(equ144)

144: 197

```
                        +       plus:sign
                        -       minus:sign
                        *       asterisk
                        /       slash
                        \       back:slash
                        &       ampersand
                        >       greater:than:sign
                        <       less:than:sign
                        =       equals:sign
                        @       at:sign
mark  ::=               .       decimal:point
                        :       colon
                        ,       comma
                        ;       semicolon
                                space
                        (       left:parenthesis, parenthesis
                        )       right:parenthesis, parenthesis
                        [       left:backet, bracket
                        ]       right:bracket, bracket
                        '       prime
                        "       quotation:mark
                        $       dollar:sign
                        !       exclamation:point
```

(equ145)

145: 11, 34, 35, 50, 56, 88, 133, 183, 186, 206, 220, 221, 225, 233, 236, 241

```
                                      letter
        name  ::=    letter           numeral
                       $                $
                                        ,
```

(equ146)

146: 54, 75

```
        name:declaration  ::= NAME  statement:name     ;
                                     procedure:name
```

(equ147)

147: 31, 102

```
        named:character:variable  ::=  named:variable
```

(equ148)

148: 140

```
        named:loop:control  ::=  named:variable  (
    control:clause   )
```

(equ149)

149: 219

```
        named:statement  ::=  statement:name : statement
```

(equ150)

150: 21, 44, 80, 84, 102, 128, 138, 147, 148

named:variable ::= simple:variable
indexed:variable

(equ151)

151: 54, 164, 215

null:declaration ::= NULL ;
BEGIN END ;

(equ152)

152: 92

null:format ::=

(equ153)

153: 219

null:declaration ::= NULL ;
BEGIN END ;

(equ154)

154: 7, 19, 22, 47, 66, 74, 76, 77, 81, 108, 123, 143, 169, 177, 189, 194, 201, 210, 214, 223, 232, 233, 243, 249, 250

number ::= numeral

(equ155)

155: 130

number:of:entries:function:call ::= NENT (table:name [index:range])

(equ156)

156: 120, 145, 154, 197

```
numeral ::= 0
            1
            2
            3
            4
            5
            6
            7
            8
            9
```

(equ157)

157: 39, 159

```
numeric:constant ::= integer:constant
                     fixed:constant
                     floating:constant
                     status:constant
                     qualified:status:constant
```

(equ158)

158: 92

```
numeric:format ::= generalized:numeric:format
                   integer:format
                   fixed:format
                   floating:format
```

(equ159)

159: 3, 18, 20, 21, 23, 31, 73, 97, 99, 102, 105, 107, 114, 127, 142, 159, 161, 175, 196, 198, 200, 202, 232

```
                                    numeric:constant
                                    numeric:variable
                                    numeric:function:call
                                      +  numeric:formula
                                      -
                                    numeric:formula
    arithmetic:operator
       numeric:formula  ::=
                                       evaluation:control
    numeric:formula

                                    formula    evaluation:control
                                    attribute:association

                                    ( numeric:formula
                                    bit:formula
```

(equ160)

160: 159

```
                                    integer:variable
       numeric:variable  ::=        fixed:variable
                                    floating:variable
```

(equ161)

161: 97, 107

```
       numeric:value:formula  ::=  [ numeric:formula ]
```

(equ162)

162: 159, 176, 247

```
                                    integer:variable
       numeric:variable  ::=        fixed:variable
                                    floating:variable
```

(equ163)

163: 63

```
order:directive ::= !ORDER ;
```

(equ164)

164: 165

```
                                          null:declaration

    ordinary:table:item:declaration
       ordinary:table:body ::=
                                          BEGIN
    ordinary:table:item:declaration         END   ;

    subordinate:overlay:declaration
```

(equ165)

165: 235

```
       ordinary:table:declaration ::= ordinary:table:heading
    ordinary:table:body
```

(equ166)

166: 165

```
       ordinary:table:heading ::=
        TABLE table:name    environmental:specifier
                                  allocation:specifier

        ; allocation:increment   dimension:list

        structure:specifier    packing:specifier

        item:description    = constant:list   ;
```

(equ167)

167: 131, 164

ordinary:table:item:declaration ::=
ITEM item:name item:description packing:specifier
= constant:list ;

(equ168)

168: 51

overlay:declaration ::=
independent:overlay:declaration

subordinate:overlay:declaration

(equ169)

169: 9, 166, 167, 182, 205, 217, 218

N

packing:specifier ::= M number

D

(equ170)

170:

actual:define:parameter
formal:define:parameter
parameter ::= actual:input:parameter
formal:input:parameter
actual:output:parameter
formal:output:parameter

(equ171)

171: 18, 39, 108

```
                                        1
                                        2
          pattern:constant  ::=         3    B    count    '
      pattern:digit    '
                                        4
                                        5
```

(equ172)

172: 171

pattern:digit ::=

pattern	pattern:digit	order
0 0 0 0 0	0	
0 0 0 0 1	1	1
0 0 0 1 0	2	
0 0 0 1 1	3	
0 0 1 0 0	4	
0 0 1 0 1	5	
0 0 1 1 0	6	
0 0 1 1 1	7	3
0 1 0 0 0	8	
0 1 0 0 1	9	
0 1 0 1 0	A	
0 1 0 1 1	B	
0 1 1 0 0	C	
0 1 1 0 1	D	
0 1 1 1 0	E	
0 1 1 1 1	F	4
1 0 0 0 0	G	
1 0 0 0 1	H	
1 0 0 1 0	I	
1 0 0 1 1	J	
1 0 1 0 0	K	
1 0 1 0 1	L	
1 0 1 1 0	M	
1 0 1 1 1	N	
1 1 0 0 0	O	
1 1 0 0 1	P	
1 1 0 1 0	Q	
1 1 0 1 1	R	
1 1 1 0 0	S	
1 1 1 0 1	T	
1 1 1 1 0	U	
1 1 1 1 1	V	5

(equ173)

173: 92

```
                              1
                              2
pattern:format  ::=           3    B      count    P
                              4
                              5
```

(equ174)

174: 63

pointer:directive ::= !POINTER pointer:formula : data:name ;

(equ175)

175: 5, 8, 67, 97, 101, 118, 121, 174, 180, 208, 237

```
                           integer:constant
pointer:formula  ::=       simple:integer:variable
                           ( numeric:formula )
```

(equ176)

176: 247

pointer:variable ::= numeric:variable

(equ177)

177: 132

precision:specifier ::= number

(equ178)

178: 221, 233

```
                                ABS
                                ALL
                                ALT             NENT
                                AND             NOT
                                BEGIN           NULL
                                FIT             NWDSEN
                                BLOCK           OR
                                BY              OVERLAY
                                BYTE            PROC
                                DEF             PROGRAM
                                DEFINE          REF
                                DIRECT          REMQUO
                                DSIZE           RESERVE
                                ELSE            RETURN
                                END             SHIFT
        primitive  ::=          ENTER           SIG
                                EQV             SIGNED
                                EXIT            SIGNUM
                                FOR             SIZE
                                FORM            STATUS
                                FORMAT          STOP
                                FRAC            SWITCH
                                GOTO            TABLE
                                IF              TEST
                                IN              THEN
                                INT             TYPE
                                ISIZE           UNTIL
                                ITEM            WHILE
                                JOVIAL          XOR
                                LOC             XRAD
                                NAME            ZAP
```

(equ179)

179: 181

```
        procedure:body  ::=     declaration
                                statement
```

```
(equ180)

180:   207

       remquo:procedure:call:statement

                                                procedure:name
       @ pointer:formula

       alternate:entrance:name
          procedure:call:statement  ::=
                                                (
       actual:input:parameter    )

                                                (
       actual:input:parameter    :     ;

       actual:output:parameter    )


(equ181)

181:   59, 75, 184

          procedure:declaration  ::=  procedure:heading
       procedure:body


(equ182)

182:   181
          procedure:heading  ::=
               PROC procedure:name   environmental:specifier
                                       data:allocation:specifier

         : instruction:allocation:specifier

         (   formal:input:parameter        :
         formal:output:parameter      )

           environmental:specifier    item:description
           allocation:specifier

         packing:specifier    [ bit:number ]

         =   +   constant     ;
             -
```

(equ183)

183: 5, 10, 53, 68, 90, 96, 101, 122, 138, 146, 180, 182, 167, 193

procedure:name ::= name

(equ184)

184: 54

```
                                  program:declaration
processing:declaration  ::=       procedure:declaration
                                  alternate:entrance:declaration
```

(equ185)

185: 184

```
program:declaraton  ::=  independent:program:declaration
                         dependent:program:declaration
```

(equ186)

186: 68, 112

program:name ::= name

(equ187)

187: 157

```
                                              status:list:name
                                              item:name
qualified:status:constant  ::=  V(            table:name          : status  )
                                              procedure:name
                                              alternate:entrance:name
```

(equ188)

188: 63

```
recursive:directive ::= !RECURSIVE ;
```

(equ189)

189: 16, 211

```
reference ::= number
```

(equ190)

190: 24, 33, 246

```
                                        <      less than
                                        =      equal
relational:operator ::=                 >      greater than
                                        >=     greater than or
   equal, not less than
                                        <>     less than or greater
   than, not equal
                                        <=     less than or equal,
   not greater than
```

(equ191)

191: 180

```
remquo:procedure:call:statement ::=
 REMQUO ( actual:input:parameter ,
 actual:input:parameter
    : actual:output:parameter , actual:output:parameter
 ) ;
```

(equ192)

192: 43

```
replacement:phrase ::= THEN   formula
                              value:formula
```

```
(equ193)

193:   207

       return:statement  ::=  RETURN   procedure:name
    ;                                  alternate:entrance:name


(equ194)

194:   77, 81

       scale  ::=  number


(equ195)

195:   63

       sets:directive  ::=  !SETS  data:name    ;


(equ196)

196:   18, 130

       shift:function:call  ::=  SHIFT ( bit:formula ,
    numeric:formula )


(equ197)

197:   25, 57, 234

                         letter
       sign  ::=         numeral
                         mark


(equ198)

198:   18, 130

       signed:function:call  ::=  SIGNED ( numeric:formula )
```

(equ199)

199: 82

```
                                        count   D   ,   count   D
    count   Z

                                        count   D   ,   count   D

                                        count   D   ,
         significand  ::=   +
                            -
                                        count   D   *   count   D

                                        count   *       count   D

                                        count   D       count   *
```

(equ200)

200: 130

significand:function:call ::= SIG (numeric:formula)

(equ201)

201: 132

significand:specifier ::= number

(equ202)

202: 130

signum:function:call ::= SIGNUM (numeric:formula)

(equ203)

203: 207

```
simple:assignment:statement  ::=  variable = formula ;
```

(equ204)

204: 175

```
simple:integer:variable  ::=  simple:variable
```

(equ205)

205: 49, 75, 131

```
simple:item:declaration  ::=
 ITEM  item:name     environmental:specifier
                      allocation:specifier

   item:description   packing:specifier

    [ bit:number ]      =   +   constant   ;
                       -
```

(equ206)

206: 90, 91, 109, 143, 208, 243

```
simple:item:name  ::=  name
```

(equ207)

207: 219

```
                                    simple:assignment:statement
                                    assignment:statement
                                    exchange:statement
                                    go:to:statement
                                    exit:statement
                                    test:statement
        simple:statement  ::=       return:statement
                                    zap:statement
                                    stop:statement
                                    loop:statement
                                    conditional:statement
                                    switch:statement
                                    procedure:call:statement
                                    direct:statement
```

(equ208)

208: 150, 204

simple:variable ::= simple:item:name @ pointer:formula

(equ209)

209: 130

```
                                              formula
        size:function:call  ::=  SIZE  (                      )
                                              data:block:name
```

(equ210)

210: 132

size:specifier ::= number

(equ211)

211: 63

```
skip:directive ::= !SKIP reference ;
```

(equ212)

212: 92

```
skip:format ::= X
```

(equ213)

213: 63

```
space:directive ::= !SPACE character ;
```

(equ214)

214: 109

```
spacer ::= number
```

(equ215)

215: 216

```
                                              null:declaration
   specified:table:body ::=
specified:table:item:declaration
                                              BEGIN
specified:table:item:declaration              END ;
```

(equ216)

216: 235

```
   specified:table:declaration ::=
specified:table:heading  specified:table:body
```

(equ217)

217: 216

```
environmental:specifier
    specified:table:heading  ::=  TABLE table:name

allocation:specifier

          : allocation:increment    dimension:list

          structure:specifier

          words:per:entry
          bits:per:entry    bit:number    entries:per:word

          packingspecifier    item:description
          packing:specifier

          [ bit:number    , word:number   ]    =
          constant:list   ;
```

(equ218)

218: 131, 215

```
    specified:table:item:declaration  ::=  ITEM item:name
item:description

          packing:specifier    [ bit:number
          , word:number   ]   = constant   ;
```

(equ219)

219: 36, 45, 112, 149, 179, 232

```
                          null:statement
    statement  ::=        simple:statement
                          compound:statement
                          named:statement
```

(equ220)

220: 5, 38, 71, 90, 104, 138, 146, 149, 232

```
statement:name ::= name
```

(equ221)

221: 187, 222, 233

```
                    primitive
status ::=          name
                    letter
```

(equ222)

222: 157, 223

```
status:constant ::= V( status )
```

(equ223)

223: 132, 224

```
status:list ::=   [  +  number ]  status:constant
                     -

        [  +  number ]  status:constant
           -
```

(equ224)

224: 54

```
    status:list:declaration ::= STATUS status:list:name
status:list ;
```

(equ225)

225: 132, 187, 224

```
status:list:name ::= name
```

(equ226)

226: 207

stop:statement ::= STOP ;

(equ227)

227: 166, 217

structure:specifier ::= P
T

(equ228)

228: 164, 168

subordinate:overlay:declaration ::= OVERLAY
subordinate:overlay:expression ;

(equ229)

229: 231

subordinate:overlay:element ::= item:name
(
subordinate:overlay:expression)

(equ230)

230: 228, 229

subordinate:overlay:expression ::=

subordinate:overlay:string :
subordinate:overlay:string

(equ231)

231: 230

```
    subordinate:overlay:string  ::=
subordinate:overlay:element
```

(equ232)

232: 207

```
    switch:statement  ::=  SWITCH numeric:formula ;
statement:name :

         BEGIN    [   +   number ]    statement   ,
                      -

         END   ;
```

(equ233)

233:

```
                              primitive
                              ideogram
                              name
                              letter:control:variable
         symbol  ::=          abbreviation
                              number
                              constant
                              comment
                              directive:key
                              status
```

(equ234)

234: 25

system:dependent:character
Most computer systems can read and write more characters than are encompassed in the set of JOVIAL "sign , The entire set that can be handled is know as the set of "characters , The "characters that are not "signs are known as "system:dependent:characters ,

(equ235)

235: 49, 51, 75

```
                                    ordinary:table:declaration
     table:declaration  ::=
                                    specified:table:declaration
```

(equ236)

236: 5, 52, 67, 90, 109, 118, 138, 155, 166, 187, 217, 251, 252

```
     table:name  ::=  name
```

(equ237)

237: 117

```
     table:variable  ::=  item:name   [  index  ]   @
  pointer:formula
```

(equ238)

238: 43

```
                                     WHILE   conditional:formula
                                     UNTIL
     terminator:phrase  ::=
                                    value:terminator
```

(equ239)

239: 207

```
     test:statement  ::=  TEST  character   ;
```

(equ240)

240: 63

```
     time:directive  ::=  !TIME  character   ;
```

(equ241)

241: 63

```
trace:directive ::= !TRACE ( conditional:formula )
    name ;
```

(equ242)

242: 130

```
type:function:call ::= TYPE ( formula )
```

(equ243)

243: 61

```
                          number
upper:bound ::=
                          simple:item:name
```

(equ244)

244: 63

```
uses:directive ::= !USES data:name ;
```

(equ245)

245: 97, 192, 246

```
value:formula ::= [ formula ]
```

(equ246)

246: 238

```
value:terminator ::=

        WHILE   value:formula  relational:operator
        variable
        UNTIL   variable  relational:operator
        value:formula
```

(equ247)

247: 5, 6, 14, 70, 203, 246

```
                    pointer:variable
                    numeric:variable
variable  ::=       bit:variable
                    character:variable
                    format:variable
```

(equ248)

248: 5, 6, 14, 70, 203, 246

```
                    named:variable
variable  ::=       letter:control:variable
                    functional:variable
```

(equ249)

249: 217, 218

```
word:number  ::=  number
```

(equ250)

250: 217

```
words:per:entry  ::=  number
```

(equ251)

251: 130

```
words:per:entry:function:call  ::=  NWDSEN  ( table:name
)
```

(equ252)

252: 207

```
                                  table:name
zap:statement  ::=  ZAP                          ;
                                  entry:variable
```

First Experimental Comm Run

(J30546) 24-APR-74 12:45; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RJC; Sub-Collections: RADC; Clerk: DLS;
Origin: <CARRIER>COMTEST.NLS;1, 10-APR-74 06:51 DLS ;